

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**TOWARDS INTERNET PROTOCOL
OVER SEAWATER (IP/SW): FORWARD ERROR
CORRECTION (FEC) USING HAMMING CODES FOR
RELIABLE ACOUSTIC TELEMETRY**

by

Stephen Paul Reimers

September 1995

Thesis Advisors:

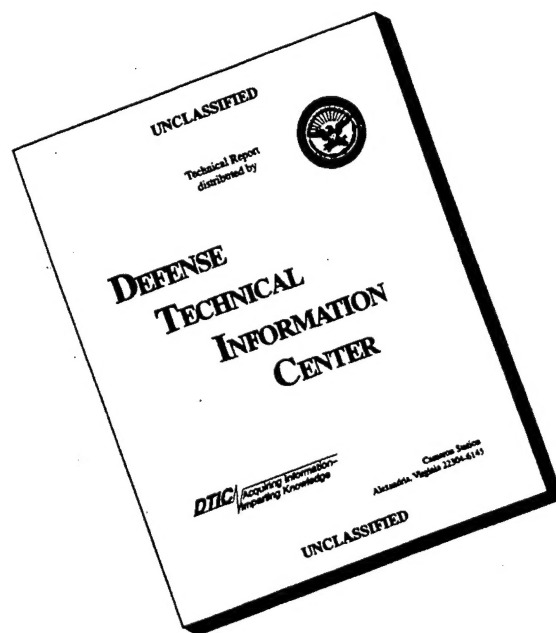
**Don Brutzman
Gilbert Lundy**

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 1

19960401 062

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGEForm Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE September 1995		3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE TOWARDS INTERNET PROTOCOL OVER SEAWATER (IP/SW): FORWARD ERROR CORRECTION (FEC) USING HAMMING CODES FOR RELIABLE ACOUSTIC TELEMETRY (U)				5. FUNDING NUMBERS	
6. AUTHOR(S) Reimers, Stephen P.					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSORING/ MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.					
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) Acoustic shallow-water data communications are unreliable. Repeated retransmission of a message received incorrectly often results in transmitter failure due to battery depletion. Internet connectivity to underwater entities via current network protocols does not exist, since most research in data communications has been focused on the physical layer and the underwater channel itself. Therefore, the problems addressed by this thesis are to find a way to increase the reliability of underwater data communications, as well as to specify the steps which need to be taken to enable Internet Protocol (IP) packets to be transmitted over an underwater channel. The approach taken was to add a Hamming forward error-detection and error-correction code to messages, thereby reducing the need for retransmission. This code was implemented in a PC-based communications system which uses commercially available diver's communication and navigation devices as acoustic modems. In a related effort, the Internet Protocol stack was analyzed for areas which required modification to enable IP to be reliably passed over an acoustic channel. The results are a low-cost reliable underwater data communications system which can easily be implemented in an AUV or underwater sensor, as well as recommendations for improved protocols required to realize true network communications in an underwater environment.					
14. SUBJECT TERMS Underwater telemetry, data communications, acoustics, forward error correction, Internet Protocol, Autonomous Underwater Vehicle, Acoustic Local Area Network, ALAN, Radio Ethernet Bridge, DiveTracker, AOSN				15. NUMBER OF PAGES 144	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL		

Approved for public release; distribution is unlimited

**TOWARDS INTERNET PROTOCOL OVER SEAWATER (IP/SW):
FORWARD ERROR CORRECTION (FEC) USING HAMMING
CODES FOR RELIABLE ACOUSTIC TELEMETRY**

Stephen Paul Reimers
Lieutenant, United States Navy
B.G.S, University of Kansas, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

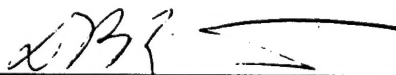
September 1995

Author:



Stephen Paul Reimers

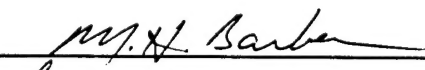
Approved by:



Don Brutzman, Thesis Advisor



Gilbert Lundy, Thesis Advisor



Ted Lewis, Chair
Department of Computer Science

ABSTRACT

Acoustic shallow-water data communications are unreliable. Repeated retransmission of a message received incorrectly often results in transmitter failure due to battery depletion. Internet connectivity to underwater entities via current network protocols does not exist, since most research in data communications has been focused on the physical layer and the underwater channel itself. Therefore, the problems addressed by this thesis are to find a way to increase the reliability of underwater data communications, as well as to specify the steps which need to be taken to enable Internet Protocol (IP) packets to be transmitted over an underwater channel.

The approach taken was to add a Hamming forward error-detection and error-correction code to messages, thereby reducing the need for retransmission. This code was implemented in a PC-based communications system which uses commercially available diver's communication and navigation devices as acoustic modems. In a related effort, the Internet Protocol stack was analyzed for areas which required modification to enable IP to be reliably passed over an acoustic channel. The results are a low-cost reliable underwater data communications system which can easily be implemented in an AUV or underwater sensor, as well as recommendations for improved protocols required to realize true network communications in an underwater environment.

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
	A. PURPOSE.....	1
	B. MOTIVATION.....	1
	C. OVERVIEW.....	2
	D. CHAPTER LAYOUT.....	2
II.	BACKGROUND AND RELATED WORK.....	5
	A. COMPUTER COMMUNICATION.....	6
	1. Communications Media.....	9
	B. HIGHER LEVEL NETWORKING.....	11
	C. THE PHYSICAL LAYER: UNDERWATER ACOUSTIC TELEMETRY..	16
	1. Characteristics of Water.....	17
	D. RELATED WORK AND STATE OF THE ART.....	19
	1. Modulation Methods.....	21
	2. Diversity Processing.....	24
	3. Protocols.....	25
	4. Noise Negation Techniques.....	26
	5. Related Non-Acoustic Research.....	28
	E. COMMUNICATION HARDWARE: MODEMS.....	29
	F. SUPPORTING EQUIPMENT: MICROWAVE RADIO ETHERNET.....	34
	G. OPERATIONAL APPLICATIONS.....	35
	1. Voice.....	35
	2. Autonomous Underwater Vehicles.....	36
	H. SUMMARY.....	36
III.	PROBLEM STATEMENT.....	39
	A. INTRODUCTION.....	39
	B. DATA COMMUNICATION IN AN UNFRIENDLY WORLD.....	39

C. TWO MODEMS DO NOT A NETWORK MAKE	40
D. A POSSIBLE SOLUTION	42
E. SUMMARY	43
IV. DATA ENCODING FOR RELIABILITY	45
A. INTRODUCTION	45
B. OVERVIEW	45
C. HISTORY OF THE HAMMING CODE	49
D. OTHER SOURCES	55
E. OTHER CONSIDERATIONS.....	56
1. Character Strings.....	56
2. Encryption for Reliability and Security: DES	56
3. Compression For Efficiency	57
F. SUMMARY	58
V. TOWARD INTERNET PROTOCOL OVER SEAWATER (IP/SW).....	59
A. INTRODUCTION	59
B. NETWORKING APPROACHES TO COMMUNICATIONS	59
C. TOWARDS IP OVER SEAWATER (IP/SW)	61
D. IMPLEMENTATION.....	63
E. SUMMARY	68
VI. EXPERIMENTAL SETUP AND RESULTS	69
A. INTRODUCTION	69
B. SETUP	69
C. HAMMING ENCODER/DECODER SIMULATION.....	70
D. COMPUTER SETUP	74
E. ACOUSTIC COMMUNICATION WITH DIVETRACKER: A MODEM....	76
F. AUV TEST TANK	77
G. THE TEST PIPE	78
H. PROBLEMS WITH DIVETRACKER.....	78

I. MOSS LANDING TEST PLAN	80
J. DATA RESULTS	84
1. DISCUSSION	84
VII. CONCLUSIONS AND RECOMMENDATIONS.....	87
A. CONCLUSIONS.....	87
1. FEC, not ARQ.....	87
2. Testing is Paramount.....	88
3. FEC is Easily Implemented	88
4. Internet Can Be Extended Underwater	88
5. Where This Is Headed.....	88
B. RECOMMENDATIONS FOR FUTURE WORK	89
1. Moving Layer Functionality Down Towards Hardware.....	89
2. Different Coding	89
3. Necessity of IETF Standards	90
4. FEC Filter to Include Selectable Encoding Algorithms	90
5. Incorporation Into NPS AUV Architecture	90
6. AOSN and Oceanographic Research	91
7. Point Sur SOSUS Array.....	91
8. Quantitatively Determine Signal to Noise Ratio (SNR).....	92
9. Implementation of DiveTracker in the WHOI ALAN Project	92
10. Protocol Analysis	92
11. Implementation of PPP	92
12. Additional Improvements to AMODEM.DC	93
APPENDIX A: CODE AND SOURCES.....	95
APPENDIX B: PROGRAMMING DIVETRACKER	125
APPENDIX C: ACRONYMS	133
APPENDIX D: RADIO ETHERNET BRIDGES	137
APPENDIX E: RETRIEVING THESIS AND SOURCE CODE.....	151

REFERENCES.....	153
INITIAL DISTRIBUTION LIST	161

LIST OF FIGURES

Figure II-1:	Serial communications channel [Blum 95].	9
Figure II-2:	TCP/IP protocol layering.	11
Figure II-3:	Internet protocol stack [Brutzman 95] [Stallings 94].	13
Figure II-4:	Internet Protocol (IP) suite layers [Brutzman 95].	14
Figure II-5:	ISO OSI Layer Protocol Functions [Stallings 95].	15
Figure II-6:	Datasonics ATM-860 acoustic modem [DataSonics 94].	30
Figure II-7:	DiveTracker diver's station DT1-D-S [Desert Star Systems 95].	32
Figure II-8:	DiveTracker surface station DT1-DRY [Desert Star Systems 95].	33
Figure II-9:	DiveTracker transducer [Desert Star Systems 95].	34
Figure II-10:	OTS Single Sideband Acoustic Comm System [OTS Corporation].	35
Figure IV-1:	Block Diagram of a digital communication system [Blahut 83].	48
Figure IV-2:	Parity bits for a 7/4 Hamming code [Stevens 93].	51
Figure IV-3:	Basic Classes of error correcting codes [Blahut 83].	54
Figure V-1:	Logical content of a IP packet.	64
Figure V-2:	Logical functional similarities between various network models and MAC level functions required in acoustic LANs.	65
Figure V-3:	Physical (Hardware) Level Transmission.	66
Figure V-4:	Logical content of a IP/SW packet.	67
Figure VI-1:	Test setup with letter labels corresponding to data sampling points.	72
Figure VI-2:	12/8 Hamming code structure.	73
Figure VI-3:	Initial test setup for error simulation.	74
Figure VI-4:	DiveTracker Setup in AUV Test Tank	77
Figure VI-5:	Data Communications in PVC pipe.	79
Figure VI-6:	Moss Landing Harbor, Moss Landing, California [NOS 92]	81
Figure VI-7:	Transducer positions for Moss Landing test [NOS 92].	82
Figure B-1:	DiveTracker test procedure.	126
Figure B-2:	Initial DIVETERM screen, DiveTracker OFF.	127
Figure B-3:	DIVETERM screen, DiveTracker ON.	128
Figure B-4:	Erase of flash memory and code download.	129
Figure B-5:	AMODEM execution response in DiveTerm.	130
Figure B-6:	PC and DT1-DRY Surface Station setup.	131
Figure B-7:	AMODEM.DC Help screen with parameter display.	132
Figure D-1:	ARLAN & AIRLAN implementation at NPS.	138
Figure D-2:	Initial Telnet Screen	140
Figure D-3:	Setting write permission.	140
Figure D-4:	Arlan screen after entering password.	141
Figure D-5:	Configuration Menu.	142
Figure D-6:	Radio Settings.	142
Figure D-7:	Ethernet Menu.	143
Figure D-8:	Indent menu.	143
Figure D-9:	Console menu.	144

Figure D-10: Airlan screen dump.	145
Figure D-11: Airlan Setup Screen	147
Figure D-12: Setting IP Address in Airlan using ABCONFIG.EXE.	148

LIST OF TABLES

Table II-1:	Communications media and common protocols.	10
Table II-2:	Acoustic modem sources and pertinent specifications.	27
Table IV-1:	Hamming Distance vs. Error Correction [Stevens 93].	52
Table IV-2:	Hamming code efficiency as function of number of bits [Stevens 93]. ...	53
Table VI-1:	Explanation of simulation parameters.	71
Table VI-2:	Simulation Results	75
Table VI-3:	AMODEM default power-on parameters [Flagg 95].	76
Table VI-4:	In-water test data.	83
Table VI-5:	In-water results for Test #13 and #16.	85

ACKNOWLEDGEMENTS

A debt of gratitude is owed to Don Brutzman, whose patience, enthusiasm, and constant encouragement never seemed to cease. Without him, this thesis would never have gotten off the ground nor completed. Without the networking insights given to me by Professor Lundy, I never would have understood digital communications. To Tom Tengdin, I owe a sincere word of appreciation for giving me a direction to go as I searched for a thesis topic, way back in the beginning. To Dr. Richard Hamming, besides a Thank-You for inventing your code, all I can say is Wow, how did you do that? To Dave Marco, I owe many fond memories of AUV missions and DiveTrackers. To Dr. Healey, a note of appreciation for buying the hardware and letting me participate in an extremely rewarding research group. I learned more in this group than in any class I ever could have taken. To Marco Flagg, fellow diver and good friend, for all the trade secrets and coding you freely gave me, I thank you. Finally, to my wife Wakana, I am eternally grateful for the care, support and feeding I received during this most challenging event in my life.

I. INTRODUCTION

A. PURPOSE

The purpose of this work is to describe the current state of underwater data communications, demonstrate how underwater communications reliability can be greatly improved using forward error correction and to outline the work required to ultimately pass Internet Protocol (IP) datagrams between two or more computers connected by an acoustic underwater channel.

B. MOTIVATION

Motivation for this study arises out of the author's interest in diving, underwater exploration and marine salvage. One of a diving supervisor's largest headaches is keeping track of a number of scuba divers in the water, since there are few ways of communicating with them as they work. One method is with underwater speaking units which transmit the diver's speech to other divers or to the surface by modulating speech onto an acoustic carrier wave. Such units require modifications to the breathing regulator to afford movement of the mouth, and often transmit breathing noises which interferes with speech. They also do not provide location data to either the diver or the surface, other than where the diver says they are, which is not always accurate. The other method is with a safety line using pull signals, which indeed provides positive control, but requires line tending and often results in tangled safety lines if multiple divers are deployed. Keeping track of underwater divers, more than any other problem, provided the spark to ignite an interest in acoustic data communication. The hardware which was able to solve the diver communication problem was also easily applied to much more significant problems in computer networking: restructuring existing acoustic protocols for reliability, and providing IP compatibility. The importance of these critical issues has not been previously recognized.

The hardware used in this thesis was originally designed for use as a diver communication and navigation device. Due to the user-configurable architecture of the hardware, it was quickly apparent that there were applications that it could be applied to that were much more complicated and closer aligned with computer science than simply communicating with divers. The study of networks and implementing them is the future of computer science. Connecting people, places and things in whatever environment they exist is quickly becoming a reality. The underwater environment is the final frontier in that exploration. The goal of extending the Internet underwater is a goal that, if realized, will facilitate great strides in underwater exploration and autonomous vehicular control.

There is another aspect to this work as well. Currently the size of the physical Internet and the corresponding volume of information content in the World-Wide Web (WWW) are growing at long-term sustained rates of 15 and 20 percent per month, respectively. The importance of gaining Internet connectivity to underwater devices for science, exploration, and collaborative research is undeniable. The potential benefits stemming from acoustic communications compatibility with IP are overwhelmingly valuable [Brutzman 94].

C. OVERVIEW

This thesis unites three separate, diverse and interrelated fields: underwater acoustics, network protocol theory, and error-correcting codes. Each of these areas has a rich history which must be appreciated in order to see what the research described in this thesis is attempting to do.

D. CHAPTER LAYOUT

The diversity of interrelated material covered in this thesis has been carefully structured to emphasize clarity. Chapter II discusses background information and previous work pertaining to computer communication, networking and underwater acoustics. Chapter III describes the problems encountered in acoustic data communication. Chapter IV outlines error correction codes: how they work, what they are used for, and (especially), how they apply to acoustic channels. Chapter V describes the TCP/IP protocol stack and

how it applies to Internet-compatible communications. Chapter VI documents the experimental setup used and experimental results obtained. Conclusions and recommendations for future work are expressed in Chapter VII. Code written for this thesis and sources for proprietary code are listed in Appendix A. A guide to programming the DiveTracker communication/navigation devices is presented in Appendix B. Appendix C is a list of acronyms used in this thesis with their English equivalents. A tutorial on setting up Arlan 620 and Airlan RF Ethernet Bridges is presented in Appendix D. Finally, Appendix E provides information pertaining to electronic retrieval of this thesis and supporting code.

II. BACKGROUND AND RELATED WORK

The subject of sound in the ocean encompasses many areas of research, from biologists studying whales and dolphins, to meteorologists trying to determine ocean temperature, to naval officers trying to locate a submarine which is thought to be in the operating area. The fact that sound travels in water farther than in air is not new, but there are many scientific and technological breakthroughs in signal processing which have occurred recently to almost make sound in the ocean a completely new area of study. In addition to the improvements and standardization of computer communications, digital communications through water is now a reality. However, water is not a perfect channel, nor does it lend itself easily to passing digital data at high speeds, which is the goal of modern computer networks. The following sections provide background in underwater acoustics and computer network theory. Recent improvements in hardware are described and compared. In addition, current documented work in the area of acoustic communications is reported in an attempt to define the state of the art of acoustic data communications.

There are significant applications for which underwater data communications are necessary. Data collection stations, such as those described for a Autonomous Oceanographic Sampling Network (AOSN) [Curtin 93] and the Modular Abyssal Benthic Laboratory (MABEL) [Luther 94] plan to rely on acoustic data links to transfer data between submerged network nodes and to direct Autonomous Underwater Vehicles (AUVs) piloting through the area. These data collection stations perform current sampling, salinity and temperature to help solve frontal dynamics problems which are at present highly dependent on limited platform (research vessel) availability. Other applications might involve teams of AUVs which are tasked with searching an area for mines and require the individual units to communicate among themselves to localize targets, provide collision avoidance, or communicate with an assist vessel on the surface. Although these applications may not sound much like a computer network, the communications networking requirements here are as applicable as in terrestrial systems.

A. COMPUTER COMMUNICATION

It is expected that the reader of this thesis already has or will obtain some knowledge of computer communications. The objective of the sections to follow is to generalize the aspects of data communications common to all networks which specifically apply to underwater acoustic communications. This will be done by approaching the relationship between a computer and its network hardware, and then describing how the hardware actually uses the medium over which it communicates. The physical medium is the point at which underwater acoustic communications diverges from common communications media. The physical medium is also the realm of acoustic modems which are tasked with the job of actual data transmission through the water.

A great deal of the insight into network communications was gained by the author through the installation of the LINUX operating system on a IBM-compatible personal computer (PC). From the first steps of reformatting the hard drive to accept an additional operating system to the actual compiling the operating system kernel, every step was outlined in articulate How-To manuals and concise walk-throughs. This experience is highly recommended for anyone desiring a deeper understanding of computer communication, operating systems, and those tired of what Microsoft or IBM have to offer in 32-bit operating systems. Most useful was the time spent on network configuration and setup. This process cements the theory learned in the classroom into practical reality. More information can be found at [University of Helsinki].

Computers communicate with one another by utilizing *networks*, structures comprised of as few as two to many links of various communication media. The media is the physical substance or material utilized to pass the computer signals over or through, and the signals are tailored for that particular medium. For example, networks that use fiber optic cable have circuitry and hardware devices to change the electrical signals the computer generates into light energy, since the fiber-optic cable transmits light rather than electrical energy. Likewise, twisted pair networks utilize wire not much different than standard telephone line, but must use specialized circuitry in each of the networked computers to convert the

digital signals generated by the computer into specific voltages and timed pulses that conform to what the other computers expect to see on the line. This lowest layer is termed the *physical* layer, and it has four important characteristics: mechanical, electrical, functional and procedural [Stallings 94].

The mechanical characteristics describe the physical connections that must be made between a computer and the network communication equipment, such as that between a computer and an external modem for communication over a telephone line, or between a computer and a network interface card (NIC) for communication over a local area network (LAN). This is usually described in a standardized connector format, such as a 9-pin connector or a DB-25 pin connector, each having certain pin numbers and purposes which are agreed upon through a common standard.

Electrical characteristics define voltage levels and timing of signals. Both the computer and the network equipment must use the same code, or communication will not occur properly. In other words, both the data terminal equipment (DTE) and data communication equipment (DCE) must be "reading the same sheet of music." The computer is labeled the data terminal equipment, since it is the source or terminus of the signals to be sent, and the network hardware (modem, network card, etc.) is the data communication equipment, as it actually performs the "communication" using the network medium.

Functional characteristics define the functions performed by the various circuits comprising the connections between the DTE and the DCE. These are usually categories such as data, timing, control, and signal ground.

Finally, procedural characteristics define a sequence of events to communicate the data, tailored to the functional characteristics specified above.

Standards introduced to completely detail these characteristics of a physical layer allow different manufacturers to build compatible components, so that products from completely separate sources can reliably and easily communicate. An example of this type of standard is the Recommended Standard Number 232, Revision C, from the Engineering Department

of the Electronics Industries Association, known as EIA RS-232C, or simply RS-232. The RS-232C standard sharply divides its functions into data functions and control functions. The data functions are assigned to the transmitter and receiver on pins 2 and 3, respectively. These are the only two pins through which data flows. All the remaining 23 pins of the standard DB-25 serial plug are control functions, so named because they carry the status or command for controlling a serial device's behavior. Additional details of RS-232 are detailed in [Campbell 87] and [Stallings 94]. An excellent hypertext discussion of RS-232 and modems is [Zinzow 94]. Another very good discussion of serial port communications is [Blum 95]. It must be made clear that the foregoing discussion of the physical layer describes the interface between the computer equipment and the communication equipment (DTE to DCE). It does **not** describe the communication which occurs between communication equipment via the network medium (DCE to DCE). Communication between DCEs is specifically designed to take advantage of the physical characteristics of the medium, whether it be the atmosphere, space, copper electrical cable, fiber optic cable, or water. Figure II-1 shows the relationship among these various components.

The serial port accepts bytes from the CPU data bus and passes bits to the modem. In doing this, the serial port can add or delete bits, depending on the coding scheme in use. At (1) the concern is with bytes per second. At (2) the concern is with bits per second, and at (3) the concern is with Baud (signal changes per second). The difference is due to the fact that the number of bits at (2) need not be equal to the number of bits (that is, bytes times 8) at (1), and the number of state changes at (3) is not necessarily the same as the number of bits at (1) or (2). Bits can be stripped going from (1) to (2): the serial port may transmit only 6 or 7 of the 8 bits in the byte. Bits can be added going from (1) to (2): the serial port can add a parity bit and stop bits.

Various methods are used to exploit the medium and carry data through it. For example, in terrestrial microwave communications, electromagnetic waves of a frequency

between 1 and 18 GHz are used to carry the encoded data from one microwave station to

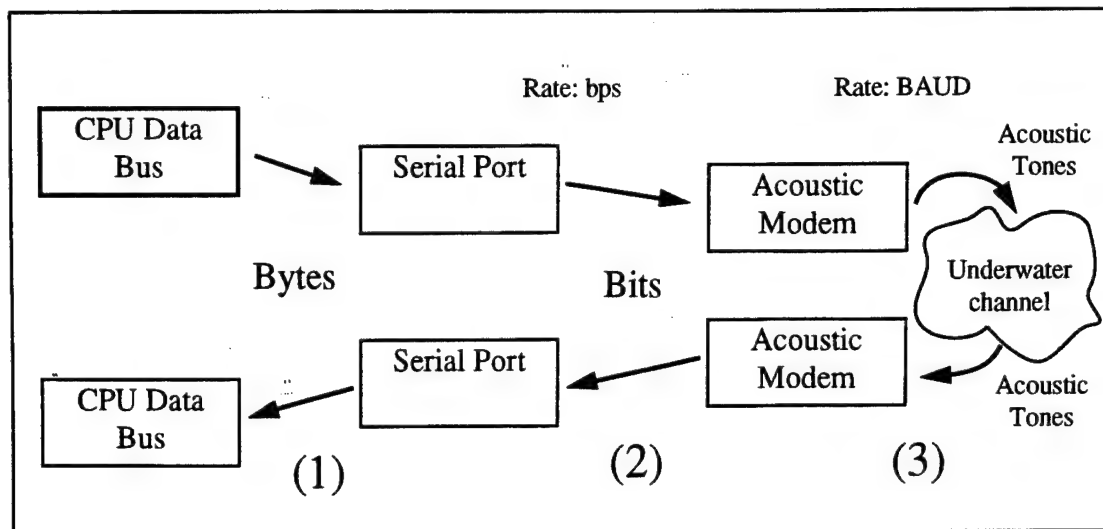


Figure II-1: Serial communications channel [Blum 95].

another. The method used to actually encode the data with the RF energy data is usually *modulation*. Modulation is the process of encoding source data onto a *carrier signal* that is compatible with the transmission media. From (2) to (3), bits may be clustered to groups that are transmitted using different encoding schemes like Frequency Shift Keying (FSK) or Quadrature Amplitude Modulation (QAM). Methods used to carry data over water will be explained in more detail.

1. Communications Media

Data communication systems in use today utilize various media over which they establish and maintain connections between individual components and networks.

Common media (and their connections) include the following:

- Twisted-pair cable (10BaseT)
- Coaxial cable (10Base2, 10Base5)
- Fiber Optic cable (FDDI, 10BaseF)
- Microwave radio frequencies (Terrestrial and Satellite communications, and microwave LAN bridges)

- Very High Frequency (VHF) and Ultra High Frequency (UHF) radio (packet radio)
- To some extent, water

Each of the above media (with the exception of water) have been exhaustively researched and exploited in order to achieve the highest possible data transfer rates and reliability possible using current technology. *Protocols* are agreed-upon sequences of data exchange that are used between communicating entities over each media. Typically protocols are optimized to take advantage of the electrical and physical characteristics of the media itself and the type of data that is to be communicated. Common media and protocols usually run on them are listed in Table II-1.

MEDIA	PROTOCOL
Twisted-Pair (10BaseT	Ethernet, Token Ring
Coaxial Cable	Ethernet, Token Ring, Token Bus
Fiber-optic Cable	Token Bus, FDDI, DQDB
Microwave RF	Ethernet, Token Ring
Water	???

Table II-1: Communications media and common protocols.

Networks today cover areas ranging from a single room in a school or business to hundreds of square miles, as in the case of wide-area network (WANs) and metropolitan-area networks (MANs). The entire group of interconnected networked computers, servers, routers, and hosts in academic institutions, commercial businesses, and homes all over the world is called the Internet. It is actually a network of networks and utilizes all of the media listed above. It is possible to easily connect almost anything to anything with the glaring exception of entities which operate autonomously in water.

B. HIGHER LEVEL NETWORKING

The preceding section describes the communication a computer has with its network hardware. That was a “middle-down” approach. This section describes a “middle-up” approach to communication. Data, in the form of electrical digital signals, travels through a series of “layers” from the user application to the actual point the computer communicates with the network. One of the most common models of internet communication, the Transport Control Protocol/Internet Protocol (TCP/IP), is described by Figure II-2, showing a basic gateway (or Internet Protocol (IP) module) associated with each Transport

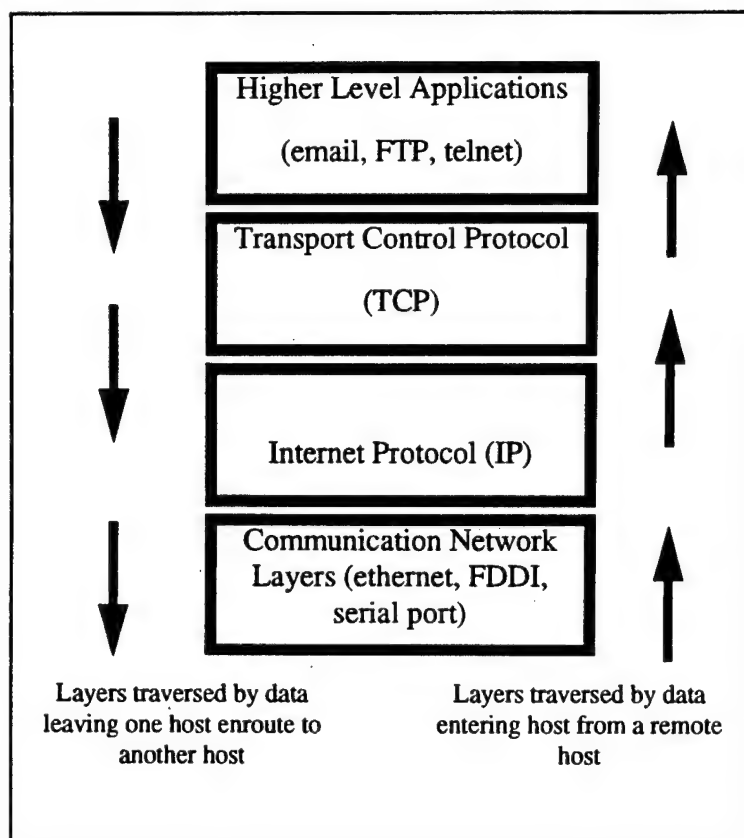


Figure II-2: TCP/IP protocol layering.

Control Protocol (TCP) packet which provides an interface to the local network. A basic “gateway” packages TCP segments inside Internet datagrams using IP and routes these

datagrams to a destination or intermediate gateway. To transmit the datagram through the local network, it is embedded in a local network packet, such as an Ethernet packet. The packet switches at the network level may perform further packaging, fragmentation, or other operations to achieve the delivery of the local packet to the destination gateway. At a gateway between networks, the IP datagram is “unwrapped” from its local packet and examined to determine through which network the Internet datagram should travel next. The IP datagram is then “wrapped” in a local packet suitable to the next network and routed to the next gateway. A gateway is permitted to break up a datagram into smaller datagram fragments if this is necessary for transmission through the network. To do this, the gateway produces a set of datagrams, each carrying a fragment. Fragments may be broken into smaller ones at intermediate gateways. The IP datagram fragment format is designed so that the destination gateway can reassemble fragments into Internet datagrams. A destination gateway unwraps the segment from the datagram (after reassembling the datagram, if necessary) and passes it to the destination TCP. An example of this communication between two hosts through the network is shown in Figure II-3.

Each of the layers in the TCP/IP stack have specific functions which the software and hardware at that level is expected to accomplish. Furthermore, each layer has a set “appearance” to layers above and below it, so that communications with each layer take place according to agreed-upon rules or *protocols*. Functions performed by each layer are outlined in Figure II-4. Request For Comments (RFC) 791 [Postel 81] is the “bible” of the Internet Protocol, and should be referred to for datagram format and protocol operation. An outstanding tutorial on the Internet Protocols is found in [Hedrick 87].

The TCP/IP model is not the only model of communications, however. The Open Systems Interface (OSI) model of the International Standards Organization (ISO) also has layers which are characterized by various protocols which pass data from one layer to another, starting at the bottom in the physical layer with the actual transmission of the 1s and 0s over the media. This layer interacts closely with the network layer to ensure error-free transmission and receipt of groups of bits, called frames, which are grouped into

packets, and given addresses which are routed by the network layer in either a

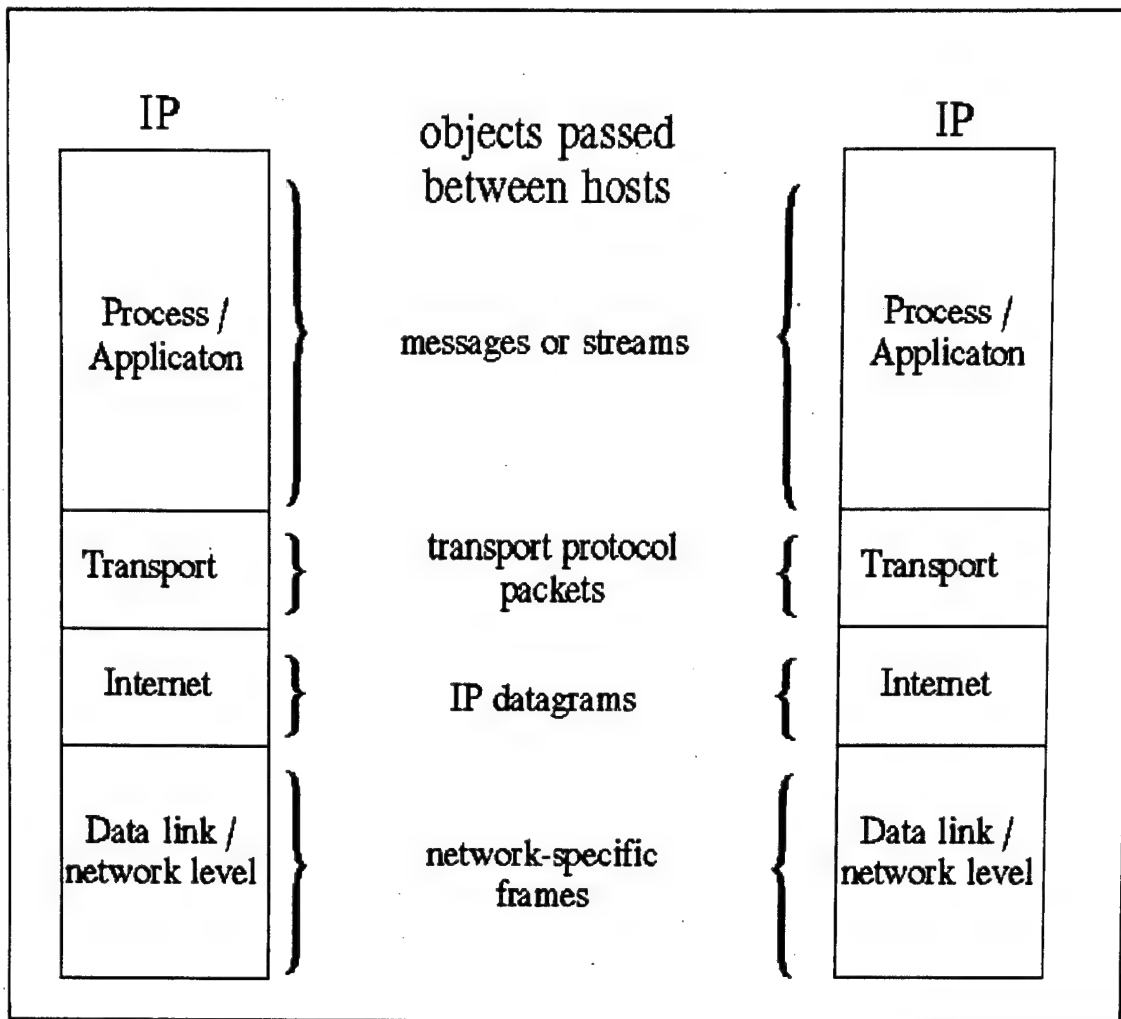


Figure II-3: Internet protocol stack [Brutzman 95] [Stallings 94].

connectionless protocol or one which is connection-oriented. The term connection here refers to the idea of setting up a dedicated path between two or more participating nodes, and keeping this path in existence until the data has been transferred and acknowledged. Connectionless protocols, on the other hand, route each packet independently of all others through the network, requiring the recipient to reorder the packets upon receipt. Lost or damaged packets, in either type of network protocol, are taken care of by the transport layer in its job of providing reliable service to the application layer, which provides the original

information to the network itself. Functions of each of the layers in the OSI model are

- **Process/Application Layer.** Applications invoke TCP/IP services, sending and receiving messages or streams with other hosts. Delivery can be intermittent or continuous.
- **Transport Layer.** Provides host-to-host packetized communications between applications, using either reliable delivery connection-oriented Transmission Control Protocol (TCP) or unreliable delivery connectionless User Datagram Protocol (UDP). Exchanges packets end-to-end with other hosts.
- **Internet/Gateway Layer.** Encapsulates packets in an IP datagram which contains routing information. Receives or ignores incoming datagrams as appropriate from other hosts. Checks datagram validity, handles network error and control messages.
- **Data Link/Network Physical Layer.** Includes physical media signaling and lowest hardware functions. Exchanges network-specific data frames with other devices. Includes capability to screen multicast packets by port number at the hardware level.

Figure II-4: Internet Protocol (IP) suite layers [Brutzman 95].

presented in Figure II-5.

One thing that sets apart the ISO-OSI system is the extensive error-checking which goes on at each layer of the process, effectively taking the data which is to be communicated and “wrapping” it with a header from each underlying layer until the message presented to the physical layer is merely a physical layer header preceding a header from each preceding layer followed by a small amount of code from the application. In other words, extreme overhead is a problem. A newcomer to the scene is the client-server model, which recognizes the ability of a terrestrial LAN to carry data at rates which are unattainable on a WAN due to the load which the ISO OSI layers put on the network node’s Central Processing Unit (CPU) while it labors at stripping and re-installing header information onto a message. In essence, the client-server model skips the network and transport layers of the ISO OSI model and concerns itself with only the application, the data link, and the physical layers to send connectionless packets between machines. Remote

Procedure Call (RPC), like the client-server model, utilizes the high data rates achievable

- **Physical Layer:** responsible for the mechanical and electrical interface to the communications media.
- **Data Link Layer:** responsible for the transmission, framing and error control over a single communications link.
- **Network Layer:** responsible for the data transfer across the network, independent of both the media comprising the underlying subnetworks and topology of those sub-networks.
- **Transport Layer:** responsible for the reliability and multiplexing of data transfer across the network to the level provided by the application.
- **Session Layer:** provides the management and data flow control services.
- **Presentation Layer:** provide a common representation of application layer data. Basically responsible for adding structures to the units of data that are exchanged.
- **Application Layer:** layer where the Goal of OSI is realized. Ultimately responsible for managing the communications between applications.

Figure II-5: ISO OSI Layer Protocol Functions [Stallings 95].

on a LAN, but does its work without the intense Input-Output (IO) that characterizes the client-server model. RPC was designed to allow a process running on one machine to call a procedure running on another machine, and for all the work to be done at the operating system level, including the actual movement of bits from one host to another. However, RPC is applicable only to situations where only one machine is communicating with another. Group communication through a broadcast medium is not allowed [Tanenbaum 92]. Of all the protocol stacks discussed, the TCP/IP stack is probably the most widely utilized due the availability of the Microsoft Windows 3.1 Windows Socket (WINSOCK). The WINSOCK is a Dynamic Link Library (.DLL) and runs under Windows 3.x, Windows for Workgroups, Windows NT, and Windows 95. It is the interface between application

programs running on PCs and the TCP/IP stack. From the WINSOCK, the information travels out to the Internet. There are literally hundreds of TCP/IP internet applications built for WINSOCK, and this alone makes TCP/IP protocol compatibility for underwater acoustic devices a goal towards which to work.

C. THE PHYSICAL LAYER: UNDERWATER ACOUSTIC TELEMETRY

Using sound in water has long been used for navigation, location, and communication. As early as 1490 Leonardo da Vinci wrote "If you cause your ship to stop, and place the head of a long tube in the water and place the outer extremity to your ear, you will hear ships at a great distance from you" [Urick 83]. This device evolved into actual steerable listening devices mounted on the hulls of ships, thereby gaining direction of the source of the sounds. In 1827, the speed of sound in water was accurately determined. The late 1800's saw the invention of transducers, devices used to change electrical energy into acoustic energy and vice versa. The turn of the century brought one of the first practical applications of underwater sound, the submarine bell, which was used by ships to determine distance from land or a lightship where a foghorn was installed. By timing the interval between hearing the underwater bell and the foghorn, they could determine how far away they were from the source of the signal. Early attempts at electronic amplification progressed into ASDIC, the British Navy's investigations into quartz transducers, and then into SONAR (SOund Navigation And Ranging), to the point where SONAR could accurately and dependably be used to determine bearing and range of a submerged target. Higher frequency sonars became common in search and location applications, due to the greater resolution afforded by the higher frequencies. However, most of these applications did not involve communication, since rangefinding and target location do not require participation from more than one entity. Not until the invention of the underwater telephone during WWII, which was much like a radio transmitter-receiver set, did actual communications take place from submarines to surface ships or to other submarines [Piccard 61].

1. Characteristics of Water

Water is an excellent working material for many applications, but carrying data, either by electromagnetic or acoustic means, is not an application that water handles nearly as well as communications media used in terrestrial computer networks. Generally, acoustic frequencies ranging from 10 to 100 kiloHertz (kHz) travel quite well in water over considerable distances, up to perhaps 20-30 kilometers. Some frequencies travel much further, even so far as to allow listening stations on the Western coast of California to hear underwater volcanic eruptions off the coast of Japan [Piccard 61]. Although the acoustic energy is able to be received at long distances, it is subject to many factors which make it less desirable for the reliable transfer of data. However, because light and electromagnetic transmissions do not pass through the water without tremendous attenuation, modulation of an acoustic signal is often the only means available for long-range underwater communication.

Sound typically travels in seawater at about 1450 meters per second (1600 yards per second), 5 orders of magnitude slower than the rate electrical signals travel in a twisted pair or coax cable. This characteristic alone differentiates water from any other media used to carry data. Noise is introduced into the medium by rain, surface action, machinery, self-noise, and biologics. Distortion of acoustic signals caused by frequency dependent absorption of segments of the signal and nonlinearities in the acoustic transducers add to the harshness of the channel. Sound wave refraction occurs due to changes in the speed of sound in water, which is continuously dependent on variations in depth, temperature and salinity. Refraction in turn changes sound propagation paths which lead to fading or multipath reception which the receiver must resolve. Multipath, arrival of a single signal at multiple times (due to parts of the signal taking a longer or shorter path to the receiver) produces intersymbol interference (ISI) and can wreak havoc on a receiver's ability to coherently gather data. Attenuation is caused by the spherical or cylindrical spreading of sound from its source and absorption of energy by the water in the form of heat. Doppler effects cause frequency shifts of the signal due to relative motion between the receiver and

transmitter, as is the case for a moving AUV. Noise, distortion, and attenuation necessitate high power levels to achieve adequate signal-to-noise ratios (SNR) for receipt of a signal. In turn, high transmit power levels create reverberation which effectively blanks the transmitter's ability to receive for a short time following transmission.

Bandwidth requirements of underwater networks and point-to-point applications have not yet approached those of conventional shore-based networks, but will be faced with limits on the order of tens of thousands of bits per second (bps) for very short ranges to hundreds of bps at moderate ranges. This is due simply to the fact that the ocean channel occupies a very narrow frequency band which is usually only 5-10 times as wide as the transmitter's peak signal rate.

[Urlick 83] provides an excellent introductory analysis of these many competing and interrelated factors. Of particular note is that variations in all these sound characteristics can occur over both short and long time scales, creating what are called *time-varying* channels. In addition, it is important to note that while these factors are present in all underwater communications channels, the amount to which they have an effect on data throughput depends to a large degree on the orientation of the channel. Vertically oriented channels are more often able to minimize the negative effects of these characteristics because of the lack of bottom and surface reflection, reduced multipath propagation, nearly perpendicular angles of thermocline penetration, and distances limited by ocean depth. Horizontal channels, particularly *shallow-water channels*, have to deal with surface and bottom reflections of acoustic energy, which destructively interferes with the transmitted pulse and causes multipath reception at the receiver. Additionally, horizontal channels create shallow grazing angles for acoustic waves which challenge the penetration of thermoclines, and broadband frequency noise is a much larger problem in horizontal channels than in vertical channels.

Through-water communications using alternative forms of energy have been demonstrated, usually with modulated coherent light systems (LASER) and have shown exceptionally high data rates are possible [Dunbar 91]. Unfortunately, the narrow

beam-width of coherent light transmission systems requires tracking systems for both receiver and transmitter to stay aligned if one or both are maneuvering. Rapid absorption of light in seawater limits the range of transmission to short distances roughly corresponding to underwater visual range. Very Low Frequency (VLF) electromagnetic (RF) transmissions have also been successfully utilized to communicate with submerged units, but the bandwidth and data rates possible using VLF equipment is far too low to consider for any practical data communications other than alertment systems.

D. RELATED WORK AND STATE OF THE ART

A good overview and survey of acoustic communication techniques is [Baggeroer 84] complemented by a underwater acoustics text such as [Urick 83]. Much of the documented work in acoustic communications has focused on physical layer signaling to achieve maximum bit rate with minimum error rate (raw bandwidth) rather than reliable throughput (effective bandwidth). Encoding of data in the acoustic signal has been well documented in the literature. Ever in pursuit of the lowest bit error rates (BER), numerous articles describe modulation techniques, all of which operate exclusively in the physical layer.

Unfortunately, that is where most of the research stops. Granted, the many simultaneous theoretical and engineering difficulties involved in getting a single slow acoustic communication link to work effectively have much to do with this lack of progress. However, much of the work surveyed is interesting, especially those describing work over long ranges. For example, [Catipovic 94] describes state-of-the-art modems capable of 10kbps over 10 km and 3 kbps at 90 km. The applicability of this work may not be as relevant to AUV communications as we would hope for, due to the high power levels required, but it does in fact lend credence to the feasibility of long range, high data rate acoustic communications. Granted, the purpose of the original work was probably not designed to be implemented in an AUV or ALAN, but if hardware advances continue as the trend has been, the complex algorithms that make these kind of results possible will be implemented in hardware and deployable. [Curtin 93], [Catipovic 93], [McMullen 94],

[Johnson 94] and [Merriam 93] all reference the Acoustic Area Network (ALAN) “presently deployed in Monterey Canyon connecting ocean bottom seismometers and current meters to Internet, enabling real-time access to *in situ* instrumentation” [Curtin 93]. This is exciting! We would hope that other work would pick up where this research left off, as there are great applications in mine warfare, oceanography, and biology for such a system. The ALAN described was unique in that it operated in the vertical channel, putting it into a different class of acoustic environment than the majority of systems which will need to be created to communicate to AUVs. It is the author’s understanding that the ALAN system was removed from service recently due to rapid consumption of batteries in the bottom nodes which rendered the system unusable. On a different note, [Zielinski 93] claims data rates of 6kbps over a distance of 3.5 km in water depth of only 15 meters, using a multipath model and DPSK transmission at 10kHz. However, these ranges were simulated using the Extend simulation package. It would be wonderful if in-water tests were documented over these ranges and frequencies. [Koelher 92] describes a system similar to the Monterey Canyon ALAN, although a much simpler arrangement of a bottom-mounted sensor communicating with a surface acoustic node via DataSonics ATM 840 and ATM 850 modems. The surface unit was connected to a laboratory on shore by Very High Frequency (VHF) packet radio as was the Monterey Canyon system which used microwave ethernet bridges. Again, this arrangement utilized the vertical channel, but this work was reported in a manner that is of use to follow-on researchers. An excellent post-deployment evaluation is presented, pointing out difficulties in hardware, software, and acoustic interference from mooring chains. This report is notable for the simple fact that many of the details involved with actual in-water tests are touched upon, again stressing that simulations are fine for laboratory use, but often data which is most useful is that which is gained through often-difficult in-water tests, and even then, caution must be taken to determine if the results can be duplicated elsewhere.

Some of the reported literature comes from industry, where claims to success are quite frequent. However, a good deal of the work reported by industry is written for and readable

by the student, as opposed to a large majority of the work written by academics which is shrouded in technicalities and couched in what is obviously "community language" unfamiliar to the uninitiated. [Garner 92] describes the history behind the Adjustable Diversity Acoustic Telemetry System (ADATS), and the work following it to create the successor of ADATS which uses digital signal processing (DSP). Data rates from 19.5 to 2500 bps over 4000 yards (depending on data rate) are claimed. If duplicable, these figures lend great promise to the ability to pass Internet Protocol over Seawater (IPOVERSW). Actual in-water tests in waters around San Diego confirmed the effectiveness of the system. It is interesting that effects of range on Bit Error Rate (BER) as predicted by the models used to design the system were not realized, introducing additional doubt into the applicability of reported work based on models and simulations alone. Approaches to dealing with the problems of acoustic communications come from a variety of directions. A brief synopsis follows:

1. Modulation Methods

Many acoustic modem designers favor non-coherent Frequency Shift Keying (FSK) and Multiple Frequency Shift Keying (MFSK) [Catipovic 93] [Coates 94] [Vukadin 91] due to the simplicity of encoding and decoding the data carried by this form of modulation. However, these non-coherent methods do not efficiently utilize scarce available bandwidth, because dispersed frequencies are often not fully utilized. Coherent modulation techniques, such as differential phase shift keying (DPSK) and quadrature phase shift keying (QPSK), which use a phase shift of 180 and 90 degrees respectively to encode a 1 or 0 compared to the previous bit transmitted [Stallings 94] have been proposed for use in AUV communications applications [Falahati 91]. Binary Phase Shift Keying (BPSK or PSK2) has proven especially useful in vertical data links. Modulation implemented by differential coding has been shown to enable receivers to solve the ambiguity of the carrier phase, and creation of transitions in the transmitted signal by scrambling have helped in synchronization at the demodulation stage [Ayela 94]. Motion of senders and receivers in

the water column as well as changes in the water itself cause severe phase fluctuations in signals being carried by the water. For this reason, [Catipovic 90] cautions against the use of coherent modulation in underwater acoustic channels (UWA) but [Stojanovic 94] reports use of receiver which performs optimal phase synchronization and channel equalization implementing PSK over ranges from 40 to 140 nm at data rates from 33 to 1000 symbols per second. Caution must be used when interpreting this data in view of the purpose of this thesis. Ability to utilize coherent modulation methods is of interest to acoustic device designers and should be considered when standards discussions take place. Additional work centered around adaptive equalization has been reported, which is a method of dealing with ISI in high-speed digital communications in acoustic channels. An advanced tutorial on adaptive equalization is presented in [Proakis 91]. Most work utilizing adaptive equalization is done with the intent of utilizing coherent modulation techniques. Adaptive equalization implementation is reported in [Tarbit 94] and [Proakis 94]. [Tarbit 94] uses actual data which was recorded from a channel 0.5 nautical miles in length and post-processed in a PC based simulation. Of note is the intention to combine the adaptive equalizer with a beamformer array to additionally reduce multipath effects. [Howe 94] describes exactly this method to achieve reported data rates of 10 kBaud over a 0.5 nm channel, but uses source signal levels of 184 dB. [Henderson 94] also reports on work centered upon adaptive beamforming, which uses an adaptive algorithm to maximize signal strength received from a desired direction while simultaneously inserting nulls in the direction from which interference is being received. This interference is usually caused by multipath. [Labat 94] reports use of jointly-optimized adaptive equalization and synchronization using QPSK. Both vertical and horizontal acoustic channels were used, and a believable data rate of 3 kBaud over a 22 meter horizontal path was achieved using an adaptive equalization algorithm implemented in a hardware digital signal processing (DSP) chip.

Although intuition tells otherwise, the most power-hungry (battery-power) operations performed by DSP modems is not transmission, it is reception. Reception of a signal calls

the DSP circuitry into action, and the complex computations designed to mitigate the effects of multipath draw substantial current as compared to the circuitry invoked in transmission [Chen 94]. Furthermore, the receiver is usually powered for longer periods of time than is the transmitter. Transmissions from submerged units are usually answers to a surface station call, or a brief data transmission, while reception is performed any time there is an acoustic signal resembling the wake-up signal for the modem. After wake-up, the receiver must listen and process until determination that the message is not for that unit, after which it can sleep. This wake-up, listen, process cycle is a large power drain on limited power reserves.

Linear frequency modulation has also been proposed as one of three modulation methods in a modem which allows the user to set the modulation method and baud rate [Ayala 94]. In this method, 1s and 0s are coded by modulating CHIRP, a spread spectrum method commonly used in radar and sonar applications. This method is easy to implement and is robust with respect to noise. However, it is limited to low-bit rate applications in which reliability is the goal, not high data rates.

Most signalling techniques in use, especially in navigation, utilize narrowband tone bursts. This type of signal is often employed because the circuitry required to transmit and receive the signal is fairly simple. Timing resolution of signals such as this depend on minimal pulse length at as high a power level as is practical. If range prohibits reception, either more power is needed or pulse length must be increased. Increasing pulse length invites multipath corruption and requires a more narrow receiver bandwidth. Use of a coded, spread-spectrum signal has been proposed as an alternative to individual channels for control and communication, giving each underwater node full use of the available bandwidth [Austin 94] [Fischer 93]. Binary phase shift keying (BPSK) and Barker codes or frequency hopping with "chips," which are broken-up parts of a single pulse, each comprised of a different signal are utilized in spread-spectrum applications. Addressing or node signal identification is accomplished by non-interfering coded signals which have good auto-correlation and poor cross-correlation properties. Direct-sequence

spread-spectrum modulation is a technique that spreads a conventional narrowband signal by mixing it with a random bit stream. The result is a dilution of the signal energy with respect to bandwidth. The spread-spectrum signal has the same energy per bit as the narrowband signal, but the power density at any one frequency is significantly lower. The signal can be spread to such a point that it is entirely below the noise level of a conventional receiver, making it difficult to detect or intercept. The receiver of a spread-spectrum signal already knows and uses the same encoded random bit stream to de-spread the signal. Through the de-spreading of the signal, the original signals are restored. This process is resistant to interference and difficult to jam.

Quadrature Amplitude Modulation (QAM) and Independent Single Sideband Modulation (ISSM) were utilized in the U.S Navy's Advanced Unmanned Search System (AUSS), a very successful underwater search vehicle built using 1970's acoustic technology [Walton 92].

It is important to note that the exact modulation method is not as important as having the same modulation method on each end of the communications channel. More important questions arise when considering reliable delivery of information. Reliable error recovery becomes the paramount bandwidth bottleneck. Additionally, coherent signal synchronization is essential to precisely determine the start of a message so that subsequent bits are properly detected, particularly in acoustic communications [Stojanovic 94]. Ordinarily coding techniques are used exclusively to improve digital signal processing performance, although [Vukadin 91] describes incorporation of Hamming codes for forward error correction (FEC) and [McCue 92] proposes the addition of FEC for reliability in protocol implementation. The whole subject of forward error correction is addressed in Chapter IV.

2. Diversity Processing

Multipath travel often causes incoherence among transmitted waveforms. One way of recovering the data contained therein is to receive the signal at more than one physical

location, where each receiver is separated by a distance determined by the frequency and pulse shape [Catipovic 91]. This is called spatial diversity. There are other forms of diversity besides this. Mitigation of multipath effects is one of the largest problems in acoustic communications. Several approaches to dealing with it have been described, including temporal, spatial, and frequency diversity. Blind equalization algorithms successfully recover phase in multipath rich "very shallow" and "extremely shallow" channels. High data rates are achieved with high level multi-level modulation schemes such as MPSK. Modulation complexity is bounded by Intersymbol Interference (ISI) and reduction of noise margins [Bessios]. Frequency diversity using MFSK is often implemented in lower-cost acoustic modems, such as are described in Table II-2

Spatial diversity equalization [Wen 94] reports progress in using multiple receivers to minimize the effects of multipath by combining the received signals, but all the research was done using models and a simulated ocean 200 meters deep and 8 miles long.

Another method to increase reliability is beamforming and adaptive equalization. Adaptive equalization and beamforming [Billon 94] take into account the transmitter, receiver and water depths, transmission range, surface and bottom reflectivity coefficients, frequency range, modulation type (MPSK and MQAM), transmit power, and sizes of transmit and receive transducer arrays to produce generalized signal-to-noise ratio and related error data rate of the digital transmission.

Equalization methods based on maximum likelihood estimation (MLE) [Feder 91] require *a priori* knowledge of the channel characteristics which are often not available. This process is also computationally intensive, which explains the numerous complexity reduction articles [Stojanovic 93]

3. Protocols

Some research into protocols between network nodes has been documented, and is centered at the medium access layer (MAC). [McCue] presents an excellent overview of the problems in medium access for acoustic devices. The send and wait and sliding window

protocols are discussed, and trade-offs between large packets and retransmit times are presented. Forward error correction (FEC) is also presented as a remedy for the problem of excessive retransmissions and will be discussed in Chapter IV. Languages for communicating between AUVs which utilize varying ASCII message formats are described in [Turner 94], and simulation results are presented. Of note is the use of ASCII in the language representation rather than machine code primitives, in order to view interactions between communicating units on a regular video monitor. Implications of this will be discussed later in Chapter IV in discussion of error correction coding. An analysis of current packet radio network (PRN) protocols forms the basis for the protocol ideas presented in [Talavage 94], but no details as to implementation are mentioned. It is assumed that results presented were gained in simulations. The need for adaptive routing in acoustic node communication protocols is presented in [Johnson 94], but this report mainly discusses a new modem design, described in [Herold 94] which is essentially an underwater computer using network routing protocols resident in a surface host. The bandwidth of current acoustic devices is usually much less than desired for underwater applications, so specialized languages are sometimes used for enhancing reliable communications in the presence of high noise and low bandwidth [Chappell 94] [Brutzman 94] [Flagg 94].

4. Noise Negation Techniques

One source of noise which an acoustic communication system such as one aboard an AUV must deal with is that which is emitted from the AUV's own motors, screws, and on-board active sonars. Fortunately, many noises emitted from an AUV can be classified and reference signals created which characterize the self-noise. Cancellation of this noise is possible using the reference signals generated previously. Two methods have been proposed: the cascade method and the multi-channel adaptive receiver algorithm. [Catipovic 94] describes this process and provides justification for favoring the multichannel adaptive receiver approach. Another method was applied in dealing with

external noise in the AUSS, mentioned earlier. The AUSS was capable of working at depths of 20,000 feet and sending compressed video data back to its support vessel in near-real-time. The AUSS was built using 1970's acoustic technology, prior to the advent of digital signal processing. One aspect of the AUSS system which distinguishes it from actual acoustic networks is the orientation of the signals passed between transmitters and receivers, as the AUSS utilized the vertical channel almost exclusively.

One of the problems which plagued the AUSS vehicle was false carrier detection, also known as double pings, in which the attenuated but reverberated uplink transmissions from the AUSS modem were mistaken for downlink transmissions from the support vessel.

Research Group	Data Rate	Modulation Method and Frequency	Basis	Error Detection/Correction	Reference
Acoustic Telemetry Group, WHOI	5000 bps	QPSK 15 kHz	14Mhz PC/XT, DOS OS, RS-232, TI TMS320C40 DSP	None	[Herold 94]
ORCA Instrumentation	20 bps (CHIRP) 200 bps (FH) 2400 bps (PSK)	CHIRP Frequency Hop (FH) (Spread Spectrum) BPSK 10-14 kHz 50-58 kHz	Motorola HC6811, 56002 DSP, RS-232	Viterbi	[Ayela 94]
Applied Remote Technology (ART) ADATS II	FSK 19.5-1250 bps	30-40.5 kHz	ART DSP & FFT cards RS-232	Unknown, if any	[Garner 92]
DataSonics	2400 bps	16 tone MFSK 13-22 kHz	Motorola 68HC11 AT&T DSP-32C	Unknown, perhaps built in now	[Chappel 94] [Merriam?]

Table II-2: Acoustic modem sources and pertinent specifications.

Research Group	Data Rate	Modulation Method and Frequency	Basis	Error Detection/Correction	Reference
Charles Starke Draper Laboratory	1200-4800 bps	BFSK 29-40.9 kHz	Intel 386 w/ i387 NPU, INMOS transputers, TMS320C30 DSP	Convolutional Coding	[Fiorillo 90]
MICRILOR, Inc.	600 Baud	Direct Sequence Spread Spectrum 30 or 100 kHz	Serial Correlation & Matched Filtering DSP RS-232	Unknown	[Fischer 93]

Table II-2: Acoustic modem sources and pertinent specifications.

The solution to this problem was to cause the submerged search vessel to periodically sleep and go into receive mode to listen for downlink transmissions. The presence of a carrier during sleep time meant a down transmission was in progress. Additionally, the acoustic link was segregated into two separate frequencies to avoid feedback of the transmitted signal into either the support ship or the search vessel [Walton 92].

Errors in imaging data also caused problems with the AUSS. Even with reliable 2400 bps communications between the AUSS and the surface vessel achieving a bit error rate (BER) of less than 3.8×10^{-6} , a one-bit transmission error resulted in the corruption of 16 pixels of compressed sensor image. The solution was to implement a technique which threw out and retransferred a section of compressed data if a check sum error was found. This technique is described by [Watson 91] and [Uhrich 92].

5. Related Non-Acoustic Research

In many ways underwater applications have parallels with space applications, including the same communication problems. Although electromagnetic radiation in the form of radio frequency energy passes easily through the earth's atmosphere and on into

deep space, it still is affected by interference, propagation delays, and attenuation. Several projects involving deep space have been documented including [Jabbari 84].

E. COMMUNICATION HARDWARE: MODEMS

Sound energy is a physical disturbance of the environment while computers communicate using electrical voltages which are understood as digital data. Modems are used to convert the electrical pulses generated by the computer into acoustic energy which is then transmitted and received via transducers.

Modem development for underwater applications has only experienced extensive academic and commercial attention. Beginning with the original "Gertrude" underwater telephone for communicating between naval ships and submarines, the technology had essentially stagnated at the analog single-sideband modulation level until digital microprocessor technology became available in the '70s and '80s. Recently, low-cost DSP technology and parallel advances in radio and telephone communications enabled the construction of highly sophisticated acoustic modems which are relatively dependable and consumed small amounts of power [Catipovic 93]. Many modems utilized in underwater network research operated at a maximum of 4800 baud, with most using rates around 2400 baud. Typical data compression was described for a commercial search system as a two-dimensional discrete (DCT) cosine transform upon whose results Huffman and run-length coding were performed using DSP on the host computer [SEACO 92]. Nevertheless propagation delays and multiplex prevented any higher data rates. Recent advances in DSP applications to acoustic modem technology has made higher-rate acoustic modems feasible. DataSonics, located in Cataumet, Massachusetts produces a family of acoustic modems, such as the one shown in Figure II-6 More information about these modems can

be found in [Datasonics 94] and [Datasonics undated]. Orca Instrumentation of Brest,

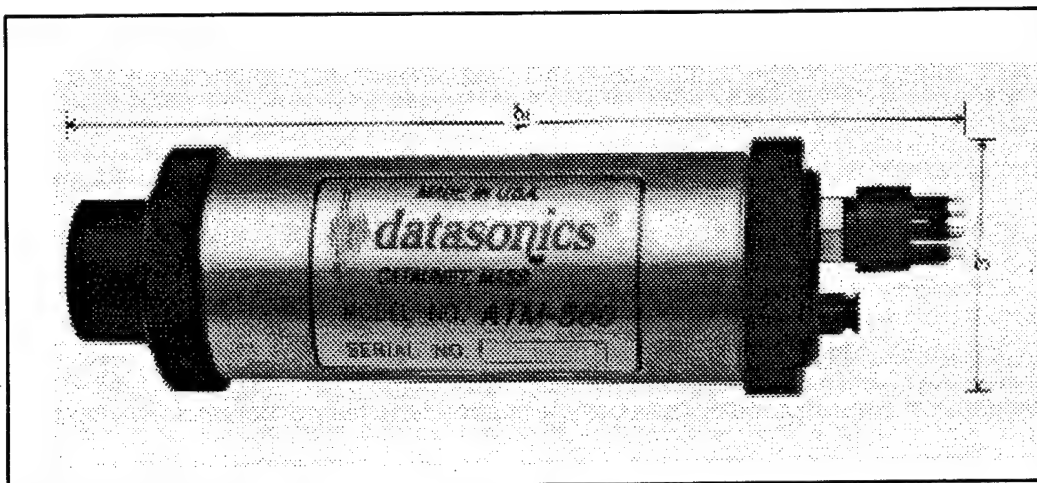


Figure II-6: Datasonics ATM-860 acoustic modem [DataSonics 94].

France is also a well-known producer of acoustic modems. DiveTracker, a multi-purpose digital communications and navigation system for divers, ROVs and AUVs combines a modem with a HC6811 microprocessor to provide long baseline (LBL) and short-baseline (SBL) navigation to both diver and surface station, as well as depth and tank pressure telemetry data to surface supervisors [Flagg 94]. Message communication is also provided by several different protocols with an RS-232 interface.

Desert Star Systems of Moss Landing, California produces the DiveTracker system. The system uses MFSK for through-water communication and navigation. The sonar modem operates in the 40-75 kHz range, and utilizes frequency-hopping techniques to exploit multipath propagation in shallow-water environments. Navigation is provided by acoustic ranging with other DiveTracker units. Omnidirectional communications are transmitted at rates from 50 to 300 bps over ranges of 300 to 3000 feet, depending on environmental conditions. The diver's unit is designed for depths of up to 1000 feet of seawater (fsw), and is capable of communicating with a surface unit and up to 9 other DiveTrackers, forming an underwater network. The surface unit is not submersible, but uses an attached submersible transducer and has the same circuitry as the diver's station.

All software for the DiveTracker system is written in C, making the unit software programmable by the user. Various software programs can be loaded into the DiveTracker's flash memory and executed from either the front panel, in the case of the Diver's Station (DT1-D-S), or from an attached PC, in the case of the Surface Station or Remote Unit (DT1-DRY or DT1-R-S). These software programs are written by Desert Star Systems and compiled using a Archimedes MS-DOS/ MC68HC11 cross-compiler. The compiled code is then loaded into one or more pages of flash memory using an accessory program named DiveTerm. The main program the DiveTracker is designed to run is a navigation and communication program named SmartDive. The divers's unit is shown in Figure II-7. Connector plugs are for sonar, RS-232 communications, battery charging, and high pressure (HP) air sensor. Dimensions are 8.5 x 3.5 x 2.16 inches. A surface station unit is shown in Figure II-8. Connector plugs are for 2 sonar transducers, RS-232 communications, and 12 volt direct current (DC) power. Dimensions are 5 x 7 x 1.5 inches. A transducer used by all the DiveTracker units is displayed in Figure II-9. Not shown is a remote station, which is built identically to the divers' station but lacks a display screen and a keypad for data entry.

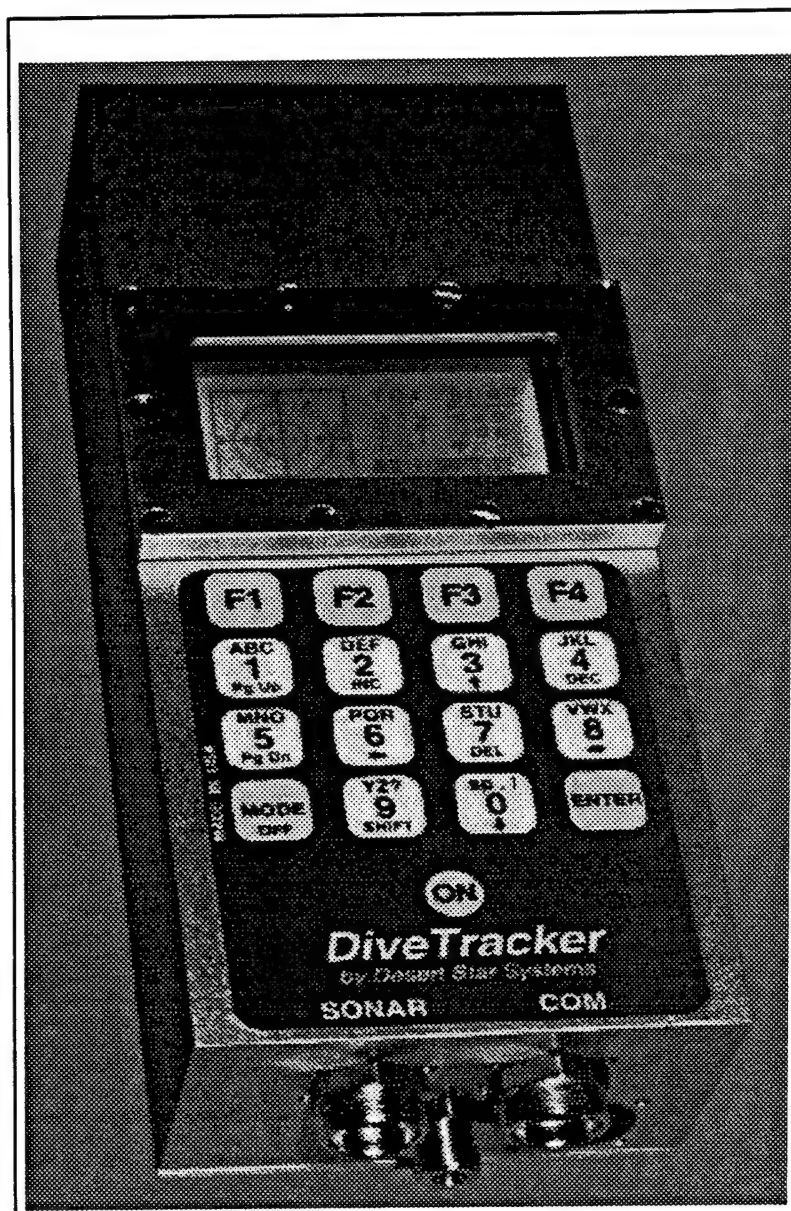


Figure II-7: DiveTracker diver's station DT1-D-S [Desert Star Systems 95].



Figure II-8: DiveTracker surface station DT1-DRY [Desert Star Systems 95].

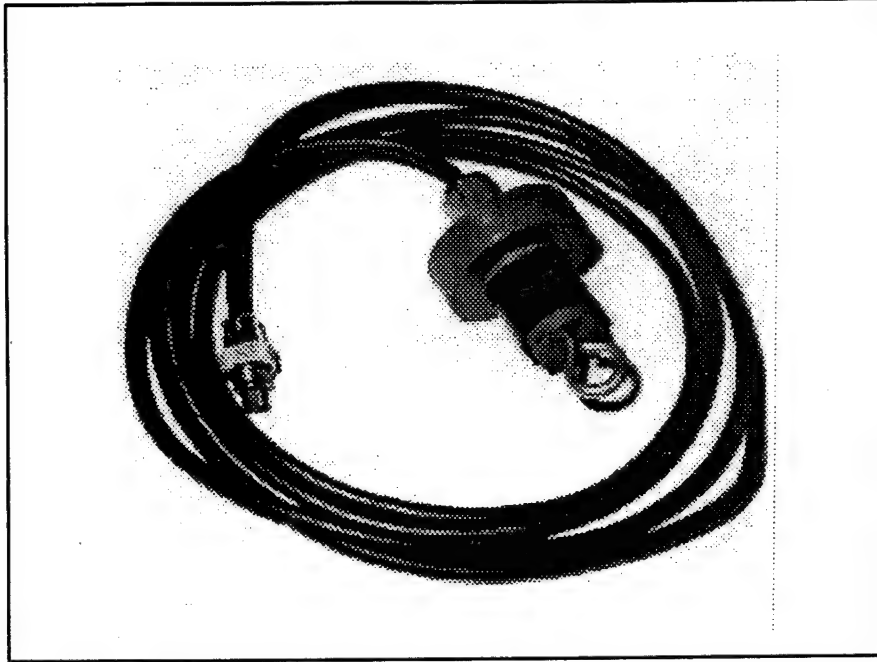


Figure II-9: DiveTracker transducer [Desert Star Systems 95].

F. SUPPORTING EQUIPMENT: MICROWAVE RADIO ETHERNET

Several methods are employed to transfer data in real-time from submerged modems to users on land, including VHF packet radio, microwave ethernet bridges, and fiber-optic cable. VHF packet radio has been successfully used although data rates and bandwidth are not as high as are available with microwave. Experimental results are described for sea-to-shore communications using an acoustic modem between a bottom station and a moored buoy with data being relayed from the buoy via VHF packet radio to a shore station, then via dial-up modem link between the shore station and the remote laboratory [Koehler 92].

Microwave is the most logical connection to buoy links and offshore acoustic LANs as it provides maximum bandwidth at minimal power. However, range is limited, often to less-than-visible ranges. Experience with two different microwave ethernet bridges is documented in Appendix D.

G. OPERATIONAL APPLICATIONS

1. Voice

Transmission and reception of voice was realized in the UQC-1 underwater telephone which was built for the US Navy to enable communications between surface ships and submarines or just between submarines. Nicknamed "Gertrude," the equipment uses single-sideband voice modulation of a 8 kHz carrier. This was a completely analog system. Newer communications systems are leaning toward all-digital signaling, incorporating many of the acoustic channel optimization methods outlined above. Of note is the work reported in [Goalic 94] which describes progress towards a digital acoustic phone capable of actual voice transmissions in real-time. An example of an operational commercially available voice communication device is shown in Figure II-10.

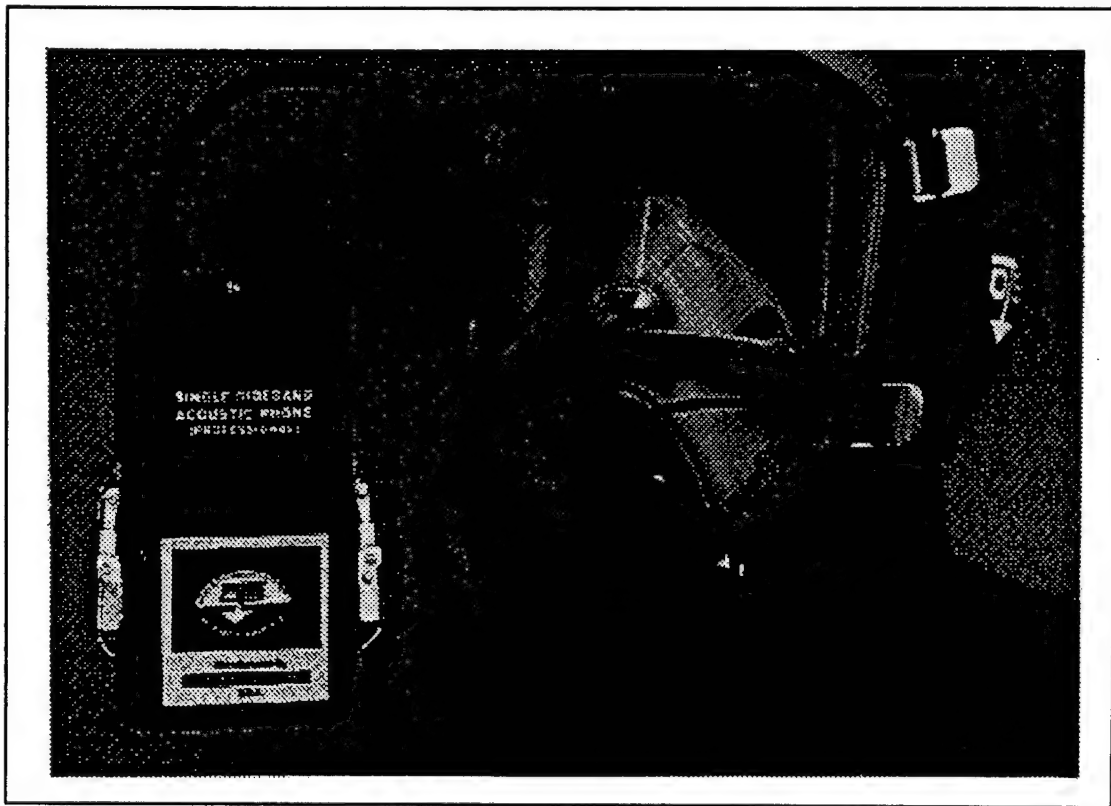


Figure II-10: OTS Single Sideband Acoustic Comm System [OTS Corporation].

2. Autonomous Underwater Vehicles

Without a doubt, the need for reliable acoustic communications at the IP level is due to the advancements being made in autonomous underwater vehicle technology. As demands for deep-diving, all-weather capable vehicles increase, more requirements are placed on communication systems to effectively transfer mission parameters, navigation information, and system status to operators and other vehicles such as remote operated vehicles (ROVs). In addition, the trend towards AUV utilization in every aspect of mine warfare continues to put a press on development of reliable, high level systems. For example, the Naval Postgraduate School *Phoenix* is currently undergoing tests with a Desert Star Systems DiveTracker communication and navigation system, described earlier. The focus of this vehicle is autonomous minefield search and localization of mine-like objects. After the vehicle returns from a search mission, the data collected by the AUV will be analyzed. If a minelike object is found, the vehicle will be commanded to re-acquire the target and place an explosive charge nearby which will be detonated at a later time. Commands and status messages will be passed to and from the vehicle acoustically while it is submerged and via microwave Ethernet link (see Appendix D) while surfaced.

H. SUMMARY

Computers communicate via networks and utilize specific protocols to transfer information from computer to computer and from layer to layer in a stack. Physical layer devices are responsible for encoding the data in order to maximize efficiency of the medium and available bandwidth. Acoustic modems are available with many different modes of transmission and reception, but reliability is still a large problem. Digital signal processing is enabling advancements in range and data rates of underwater devices, but there is much work yet to be done in this area. Notable sources and current work was reviewed, and there exists a large amount of documentation that is not only written at a very high language for the neophyte, but much of the work is reported on by numerous people without anyone duplicating the work for verification purposes. The focus of underwater

data transmission is most strongly directed at control of AUVs, but acoustic networks and commercial applications such as diver support and oceanography data collection also require reliable, long-lasting efficient units to communicate.

III. PROBLEM STATEMENT

A. INTRODUCTION

This section addresses specific problems in underwater acoustic communications which have not been a point of focus in the reported literature, yet will greatly limit the effectiveness and reliability of underwater devices. These problems center around the fact that most underwater devices have limited power supplies, since most are battery operated, and thus, communications protocols must be designed with this limitation in mind. Methods of protecting against the problem of battery depletion due to excessive retransmissions are available and is addressed. Further, a large majority of the reported work has focused on point-to-point communications, from one acoustic device to another, not on networked communication in a broadcast environment. One reason for this is lack of compatibility among acoustic devices. Another is the difficulty in specifying a protocol that will work with the harsh conditions and extreme propagation delays for which acoustic environments are well known.

B. DATA COMMUNICATION IN AN UNFRIENDLY WORLD

In an underwater environment, an autonomous entity will often have difficulty passing a message reliably to its intended receiver. Common protocols utilized in terrestrial networks depend on an acknowledgment/resend mechanism: data not received is not acknowledged, causing the sender to resend the data after a preset time period, or the message is received in error, causing the receiver to send a Not Acknowledged (NAK) to the sender, who in turn resends the required data. The problem with this common approach is that an underwater channel may have so much interference, noise, or attenuation that the message is not received in its entirety without error. Under this protocol, the sender will continue to send and resend, until its power resources have been depleted. Unfortunately, autonomous underwater entities are exclusively dependent on batteries, and transmission requires substantial power which is in limited supply. Ultimately, the sender might attempt

to resend the message until it has depleted its power supply, and then it "dies." This is called the "retry until you die" syndrome. What is needed is a way to allow the receiver to correct the messages received in error in order to reduce or eliminate the need for retransmission.

C. TWO MODEMS DO NOT A NETWORK MAKE

Currently there are many different acoustic modems used by commercial firms and research groups, as shown in Table II-2. Typically devices built by different teams are completely incompatible, both on the wet side (in-water acoustics) and dry side (upstream computer interface). Acoustic incompatibilities are understandable given the difficulty of adequate underwater communications among diverse acoustic environments. Nevertheless the many dissimilar upstream interfaces are often made to similar computers for similar reasons. Apart from the standard RS-232C interface (discussed in Chapter II), little thought has gone into design for compatibility. It seems obvious that the goal of underwater communications design personnel ought to be connectivity with the global Internet. In so doing, any user might then be able to access information contained on an underwater entity in near-real-time while at a physical location far removed from the ocean itself, using existing Internet protocols. These are the same protocols which allow applications such as mail, FTP, and telnet to be run on any computer which is connected to the Internet. Internet connectivity means that a scientist or student, (assuming they have the proper authority), might initiate or modify a mission profile of the NPS AUV while conducting a mission in Monterey Bay, merely by sending electronic mail to the AUV. Standardized communications mechanisms already exist for one-to-one and several-to-many delivery of imagery, audio, video and positional information. Adding Internet Protocol (IP) compatibility and reliability to current acoustic communications protocols can provide all of these capabilities and extend the Internet underwater. From a networking perspective, current acoustic communications devices only provide low-level functionality at the physical layer and the data link layer. The key to making Internet connectivity a reality is to analyze the functions of the current Internet protocols, and extract from them the tasks

that must be accomplished at each layer. These tasks must then be applied to a protocol which is created specifically for an underwater environment which uses acoustic communications. The addition of higher layer protocols for reliable transport and applications connectivity can augment existing acoustic communications devices in order to be compatible with the IP suite.

Users merely wish to send and receive messages via computer interfaces. The precise split between the functionality embedded in the acoustic device and the functionality performed by the host computer varies greatly among documented applications and available devices. Given that the commonality shared by various devices is much greater than the differences, there exists tremendous potential for improved interoperability among acoustic telemetry systems.

The methods used to logically partition the functionality of encoding and decoding for data and signal processing can also vary widely. Many inefficiencies and incompatibilities are introduced if network considerations are ignored. When multiple layers are intended to work together, a networked approach provides a set of protocol layers which logically and efficiently splits functionality requirements into a consistent hierarchy. This provides interoperability, efficiency and appropriate degrees of abstraction and simplification.

An example is in order here. Ethernet is a common LAN technology. Different computers with different hardware interfaces can each communicate over the medium of a shared cable. Although many signaling schemes are possible on the shared wire, a single standardized protocol defines the hardware signaling to be performed by various network interface cards (NIC) . Similarly, at higher levels of abstraction in the computer, an application program cares little if the underlying physical medium utilizes Ethernet, Fiber Distributed Data Interface (FDDI) or even Serial Line Internet Protocol (SLIP). Communication packets are still sent down to the hardware interface, and in each case network protocols ensure compatibility despite different hardware and software. Thus inconsistencies between key sonar parameters such as frequency, signalling and low-level coding need only exist at the physical and data link layers of a network hierarchy.

Transducer and signalling hardware components might be replaced without requiring any changes to the interface software above the physical layer if, for example, a new modem becomes available or a different encoding scheme is implemented in the modem.

The timing delays and ambiguities introduced by the slow propagation rate of sound in water play a dominant role in the selection of a communication protocol. Timing characteristics of the channel determines more what protocols cannot be used than which ones should. Synchronization between transmitter and receiver is key to making many coherent modulation transmission schemes work. Commonly accepted terrestrial network access protocols and physical level protocols simply do not work due to the propagation delay and unreliability of the acoustic channel. At the forefront of the acoustics problem is the lack of precise data for local environments, in addition to the non-stationary condition states which characterize them.

D. A POSSIBLE SOLUTION

Defining mission-oriented languages for acoustic communications with AUVs are useful as a way to improve clarity, improve robustness, and reduce bandwidth requirements through concise syntax. Nevertheless, such approaches do not provide a generalized solution. An optimal communications methodology is data-format independent, i.e. the acoustic channel can pass arbitrary binary data. Such a capability includes compatibility with 8-bit ASCII text which is easily read by both machines and humans. Ordinarily text is the most practical and dependable format for acoustic data transmission to underwater electronic devices such as AUVs and remote sensors. A simple general and sufficient telemetry scheme is to start each message with a reserved keyword, followed by zero or more corresponding parameters [Brutzman 94]. Numerous efficient compression and encryption schemes exist which can further reduce the size of any message to an information-theoretic minimum within acceptable computational overhead costs [Brutzman 92].

E. SUMMARY

It is proposed in this thesis that the way to solve the “retry until you die” problem is to reduce retransmissions of messages by forward error correction, a method of data encoding which gives the receiver the ability to correct data received in error up to a preset bound. In addition, it is proposed that by analyzing the existing Internet Protocol stack and determining what functions each layer performs in a terrestrial network, layer functions for an underwater acoustic network could be identified. Once layer functions are mapped out, the path is clear for standards organizations to begin building an acoustic network standard based on the Internet Protocol.

IV. DATA ENCODING FOR RELIABILITY

A. INTRODUCTION

Forward error correction has been proposed as a way to increase reliability of underwater acoustic devices. In this section, various methods of error detection and correction will be discussed. An in-depth look at Hamming error detection and correction codes is presented. Following that, differences between block and non-block codes are defined. Reasons for implementation of ASCII-based communications systems are given. Finally, data encryption for security and data compression, which are two related subjects are briefly mentioned in view of the fact that each process is designed to modify the data in different ways in order to meet users' needs, but they can be made to work together with forward error correction to increase reliability and security.

B. OVERVIEW

Communication systems connect data sources to data users through channels. A telephone line, a microwave link and a coax Ethernet link are all channels, as is the path of acoustic energy through the water from a sender to a listener. In every channel, noise attenuates and corrupts the data carried by the medium. Errors in data communications are produced in one of two ways: static events, whose behavior and existence are well known, and transient events, which occur randomly. Transients are more difficult to deal with since they cannot be anticipated. Most common transient events are impulse noise or burst noise, which can be described as periods of disrupting noise bounded on either side by periods of no noise [Campbell 87]. An excellent survey of noise in the acoustic environment is [Urick 83]. One method used to deal with errors is retransmission of the message. Retransmission of the message received in error is utilized in systems where the propagation times are minimal and there is no cost to retransmit the message. Ethernet LANs utilize retransmission extensively because the above caveats fit the Ethernet protocol well. As discussed in Chapter III, transmission of an acoustic signal requires substantial power

which is in limited supply, and lack of an acknowledgment can cause the sender to deplete its batteries.

There is a way to solve this problem, and it is the method utilized by remote robots which inhabit an equally unfriendly environment: deep space. Deep-space probes such as Voyager are subject to the same challenges as autonomous underwater entities: critical dependence on limited power reserves, huge signal propagation delays, and extremely low signal power levels vulnerable to noise and attenuation [Walker 94]. A data-encoding method known as forward error correction (FEC) can be used to provide the receiver with enough redundancy that the contents of the message can be reconstructed even if it is received with significant errors, preventing a lengthy transmission request and subsequent power-robbing retransmission of the message. Since this correction occurs on the receiver side where the message has been forwarded, it is called forward error correction. The converse situation is when errors can only be eliminated by resending the message from the sending (backward) side. Thus a (NAK)/retransmission sequence constitutes backward error correction.

Most ACK/NAK schemes operate on simple error detection, the process of checking the incoming data for corruption and inverted bits. This is usually done by a simple parity checking scheme, where blocks of data are summed modulo-2 and a parity bits for either even or odd parity is appended. In this method, no correction is available. The parity bits simply let the receiver know there is an error somewhere in the data. A more advanced method is cyclic redundancy checking (CRC) which works as follows: Given a k -bit frame or message, the transmitter generates an n -bit sequence, known as a *frame check sequence* (FCS), so that the resulting number, consisting of $k + n$ bits, is exactly divisible by some predetermined number. The receiver then divides the incoming frame by the same number and, if there is no remainder, assumes that there was no error. This method also only lets the receiver know if there were errors; it does not let the receiver know where the errors were or how to fix them.

A better approach is to not only allow the receiver to detect errors, but also to correct them, up to a limit determined by the algorithm used to encode the data. Algorithms used to encode messages for this purpose are *error-correcting codes*. The key to forward error correction (correction at the receiver) is redundancy. Indeed, the simplest error-correcting code is simply to repeat everything several times. If, for example, it is anticipated that there will be no more than one error to occur in transmission of every three bits, then repeating each bit three times and using “majority vote” at the receiving end will guarantee that the message is heard correctly (e.g. 111, 000, 011, 111 will be correctly heard as 1011). In general, n errors can be compensated for by repeating things $2n + 1$ times. However, this is inefficient, and there are better ways of adding redundancy to the data than to repeat everything.

Figure IV-1 shows a block diagram of a digital communication system that encodes transmitted data with additional information prior to transmission. Data enters the system from the source, and is then processed by a source encoder designed to represent the source data in a more compact manner. An example of this is the serial output of a PC communicating with an AUV at sea. A suitable bit rate for synchronous transmission underwater might be 4800 bps, which means each bit has a duration of 208.5 μ s. Fourier analysis shows that a continuous stream of alternate 1’s and 0’s requires a wider bandwidth than any other combination of bits [Falahati 91]. Since there is a good possibility of this happening, the bandwidth requirements for this case must be satisfied. If the maximum bit rate is 4800 bps, the minimum bandwidth required is 2400 Hz. The source encoder samples 2 bits at a time and produces a four-level stream of data symbols $\{s_i\}$. Each level corresponds to a different bit combination, such as the integral symbols 00, 01, 11, or 10. Since each level has a duration of 2 symbols (417 μ s), the symbol rate (2400 Baud) is half the bit rate and the bandwidth required is halved to 1200 Hz. The output of the source encoder is the *source codeword*. The source codeword is then processed by the channel encoder which produces another sequence called the *channel codeword*. The channel

codeword is a new, longer sequence of symbols that has the information redundancy discussed above built into it.

The modulator then converts each symbol of the channel codeword into a corresponding analog symbol from a finite set of possible analog symbols. The analog symbol string is then transmitted through the channel. Noise and other types of interference in the channel alters the analog signal so that it now looks different from what the modulator actually transmitted.

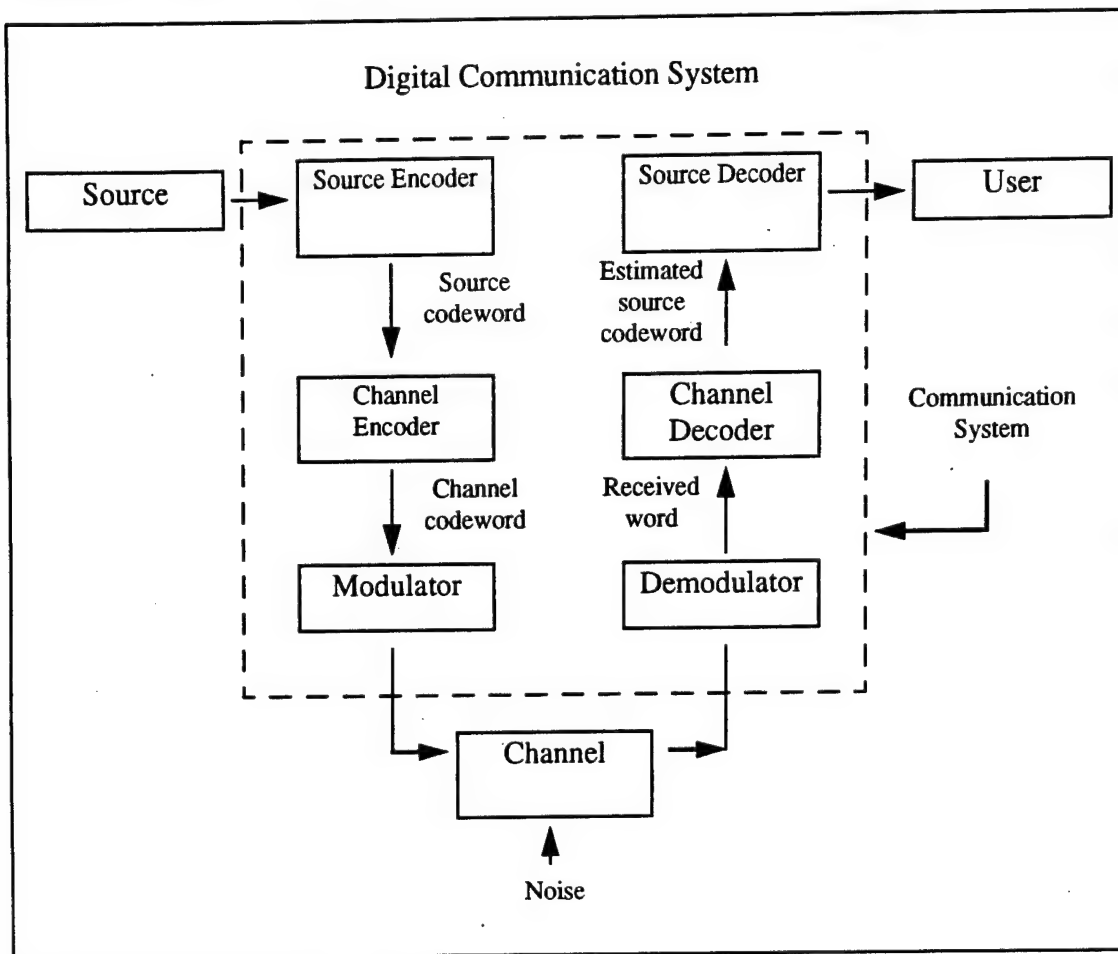


Figure IV-1: Block Diagram of a digital communication system
[Blahut 83].

The demodulator converts each received channel output symbol to one of the channel codeword symbols. However, each demodulated symbol is a estimate of the original symbol, because the demodulator makes mistakes due to channel noise. The demodulated symbol sequence is called the *received word*. Due to errors in the channel, the received word often does not look exactly like the channel codeword.

The channel decoder then uses the redundant information in the channel codeword to correct the errors in the received word and produce an estimate of the source codeword. As long as all errors are corrected, the received codeword matches the original source codeword. The source decoder performs the inverse operation of the source encoder and produces an output for the user [Blahut 83]. As long as the errors are within the capability of the error-correcting code to handle, no retransmission is needed. Thus this system has achieved the goal of reducing retransmissions.

C. HISTORY OF THE HAMMING CODE

While at Bell Laboratories in 1948, Claude Shannon proved that any communication channel has a number C (in units of bits per second) associated with it which is the capacity of the channel [Blahut 83]. This number is significant in that whenever the transmission rate R (in bits per second) required of the channel is less than C , it is possible to design an error-control code whose probability of output error is as small as is desired. Unfortunately, Shannon did not specify exactly how to build these error-control codes. He just proved that they existed.

Adding extra bits, called parity bits, to a message provides the ability to detect and correct errors. The extra bits are the redundant information mentioned above. The goal of adding parity bits is to increase the number of possible code words and define a valid and an invalid set. For example, a three-bit data word has 8 possible combinations which are all valid words. Changing one bit in any of the 8 words produces another valid but erroneous word. Addition of a fourth bit now produces 16 available combinations, 8 with an even number of 1s and 8 with an odd number of 1s. Either the even or odd set may be used as

the 8 viable codewords, but in either set, changing one bit removes that word from the set of correct combinations. However, changing 2 bits will now produce another word in the set of correct words. The number of bits which must be changed to produce another valid codeword is called the Hamming distance, named after Richard W. Hamming, who at that time also worked for Bell Laboratories.

Parity bits added to a two-dimensional matrix of bits, such as are found in a magnetic tape, became known as Vertical Redundancy Check/Longitudinal Redundancy Check (VRC/LRC). This method of error detection/correction enabled correction of single bit errors and detection of double bit errors, and became known as a "double-detection, single correction" code. The location of the bit in error was determined after reading the tape into a buffer by computing the parity across the matrix for each row, then computing the parity down the matrix for each column. The row in error (incorrect parity) and the column in error "point" to the bit in error. Assuming only one error had occurred, inversion of that identified bit corrects the block of data and hence the message is corrected.

Hamming error correcting codes arose from a desire to take humans out of the loop when checking early computers for errors [Hamming 86]. Hamming codes redundantly encode information in an optimally efficient scheme suitable for a variety of error probability distributions, over intervals and frequencies specified by the code designer. Error detection and correction can be achieved at any degree of reliability desired, at a logarithmically increasing bandwidth cost of additional parity bits and a linearly increasing computational cost of decoding. The codes were designed in the years following Shannon's work, and were able to detect and correct errors by producing a *syndrome*, i.e. the "address" of the bit in error. Hamming codes add parity bits to a message in a specific pattern, such that the parity bits have specific bit positions reserved for them. Each parity bit "covers" a domain of bit positions. For example, Figure IV-2 shows a Venn diagram which graphically represents this idea for a seven-bit word which has four data bits and three parity bits. Each of the 3 parity bits is computed from the values of the bits in the positions which that parity bit "covers." Modulo-2 addition (binary addition without carry) is used to

compute the sum and a parity bit is selected which makes the parity for the group even. The same process is repeated for each parity bit, thereby creating a 7 bit word. This is the encoded word which is sent over the communication channel. Redundant data has been added to the word, so that if no more than one bit in the seven is inverted due to noise, the receiver will be able to detect and correct it. The receiver does this by placing the received word into a buffer and extracting out the parity bits in the form of a 3 bit word. This word

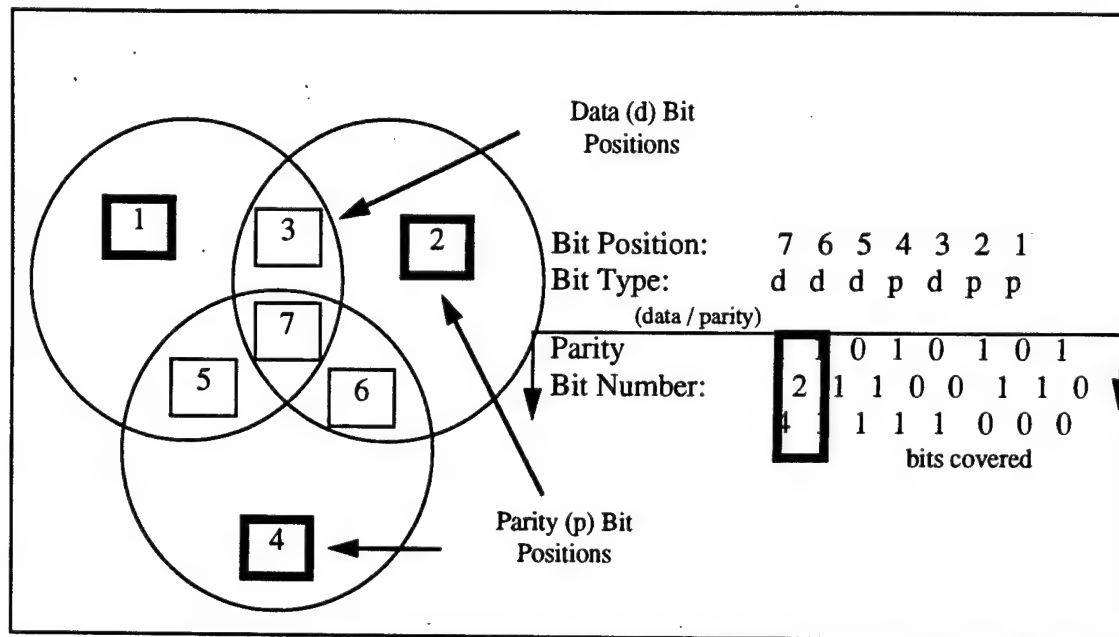


Figure IV-2: Parity bits for a 7/4 Hamming code [Stevens 93].

is held in a temporary buffer until the next step is completed, which is done when the receiver takes the data bits from the received word and recomputes the parity bits, just as the transmitter did to send the word through the channel. The new parity bits are then *X-ORed* (mod-2 addition) with the parity bits being held in the temporary buffer, and if the result does not equal zero, the received message has an error at the bit location specified by the result of the X-OR. This erroneous bit is then inverted, and the message, minus the parity bits, is sent out of the communication system. An example is outlined in [Stallings 93].

The Hamming code produced with four data bits and three parity bits has only 16 valid combinations, but with seven bits total, there are 128 various ways these seven bits can be arranged. This leaves 112 (128 - 16) combinations of bits that are not words. Each of the 16 valid words are spread evenly throughout the invalid 112, putting some “space” around each of the valid words. In order to change one valid word to another, a certain number of bits must be inverted. This number, as mentioned before, is the Hamming distance. In order for the code to be able to properly correct errors in a word, the number of errors must be less than half the minimum Hamming distance of the code. For example, in a 7/4 Hamming code, the Hamming distance is 3. Half of this is 1.5, which means that only one error per word can be corrected. However, 2 errors will be detected but there is no way to determine the original valid codeword because the number of errors is more than half the Hamming distance.

The Hamming distance between two valid codewords can be increased by adding more parity bits. Adding one extra parity bit to the 7/4 code makes it an 8/4 code and the corresponding Hamming distance is increased from 3 to 4, as shown in Table IV-1.

Minimum Hamming Distance	Error Detecting / Correcting Capability
1	None
2	Single Error Detection
3	Single Error Correction OR Double Error Detection
4	Single Error Correction AND Double Error Detection OR Triple Error Detection
5	Double Error Detection

Table IV-1: Hamming Distance vs. Error Correction [Stevens 93].

addition, more parity bits means more data bits can be “covered,” increasing the efficiency of the code, as shown in Table IV-2.

The family of codes that the Hamming codes belong to is the *block codes*. Other members of the block code family include Reed-Solomon and Bose-Chaudhuri-Hocquenghem (BCH) codes. Another family of codes also exists which are based on probabilities of errors occurring. Studies of encoding and decoding from a probabilistic point of view led to the concept of sequential decoding. Sequential codes introduced non-block codes of indefinite length, which are represented by a tree and can be decoded by searching the tree. The most highly structured (and therefore the most useful,)

Hamming Code Dimension	Number of Parity Bits	Number of Data Bits	Code Word Length	Percent Redundancy (parity/length X 100)
7/4	3	4	7	43
15/11	4	11	15	27
31/26	5	26	31	16
63/57	6	57	63	10
127/120	7	120	127	6
255/247	8	247	255	3
511/502	9	502	511	2
1023/1013	10	1013	1023	1

Table IV-2: Hamming code efficiency as function of number of bits [Stevens 93].

of these codes are called *convolutional codes*. Convolutional codes are generated by linear shift-register circuits that perform a convolution operation on the information stream. The codes are usually decoded by a method called the Viterbi algorithm. Both of these methods are summarized in Figure IV-3. Some of the articles referred to in this thesis documented work which utilized convolutional coders at the modulation level and Viterbi decoders at the receiver end. [Blahut 83] proposed that both block codes and convolutional codes might be mixed by nesting the codes. A simple convolutional code with a Viterbi decoder will do fairly well at correcting occasional errors in a long binary data stream, but too many errors

occurring closely together will be decoded into an error burst by the Viterbi decoder. A Reed-Solomon decoder on the outside of the convolutional code can be used to correct these bursts. This technique of nesting a Reed-Solomon code with a weak but simple convolutional code has been shown to be effective in a channel that demonstrates a Gaussian noise distribution. This concept, also known as code concatenation, was also presented in [Proakis 91] and results in greater bandwidth efficiency than ordinary methods for a given level of performance. This is significant in underwater acoustic communications

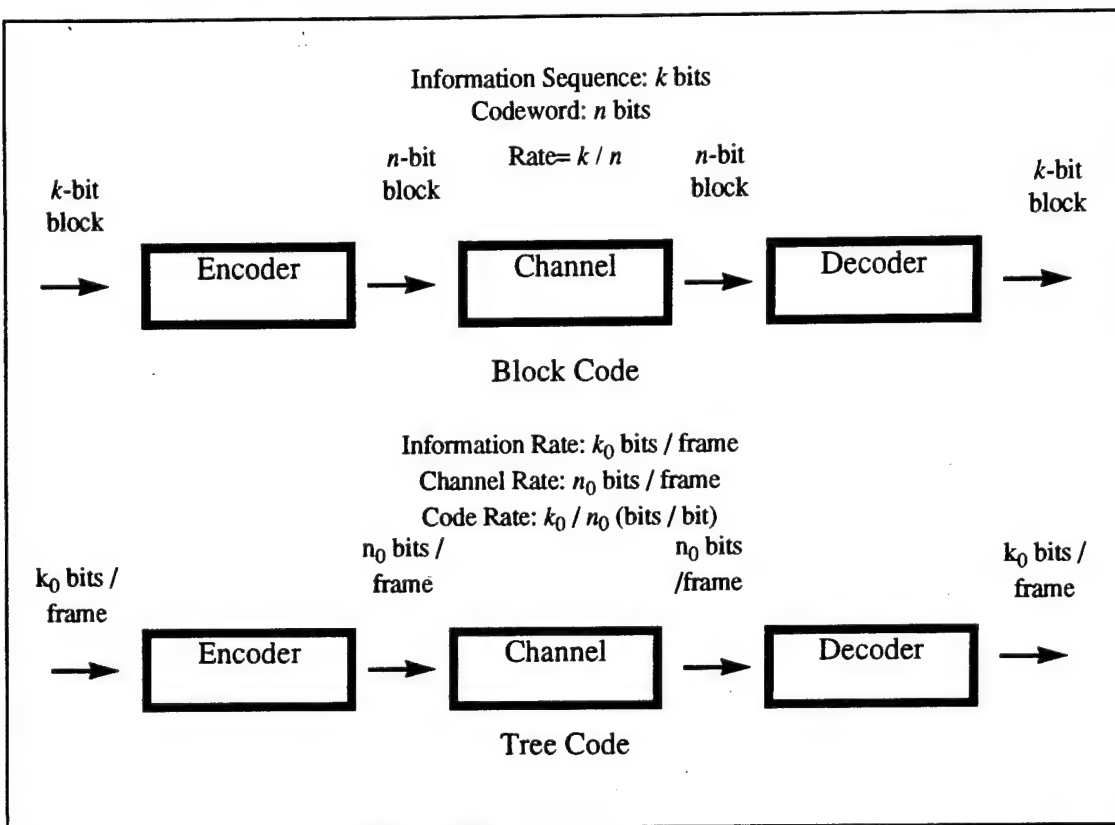


Figure IV-3: Basic Classes of error correcting codes [Blahut 83].

because many times, noise in an acoustic channel does much more damage to a data signal than to cause occasional bit errors. Some broadband, continuous noise sources can mask or delete entire blocks of data, causing the receiver to receive nothing, instead of a damaged data stream. In other words, the acoustic channel is not easily characterized as specific

noise distribution, but each code must be optimized for the particular conditions existing at that location and time of communication. Improvements to coding must be based on examining the noises present in the environment and acoustics. This problem of lost or non-existent data is not fully addressed in many of the FEC codes available. Most of the codes researched are able to deal with bit inversions, but seldom are dropped or missing chunks of data able to be overcome. Synchronization often depends on continuity of data streams, and interruption of the stream in an error correcting routine often means retransmission is mandatory. Unfortunately, the underwater acoustic channel is marked by this characteristic.

Future work in the use of FEC and error-correcting codes needs to focus on noise distribution affecting specific acoustic devices so that other codes can be applied that are better matched to the noise characteristics of the environment. For example, noise due to biologic sources is on a much shorter time scale than noise due to a motorboat passing nearby. Reed-Solomon and other codes are likely to improve error correcting performance in different scenarios. Nevertheless, computational complexity and the possibility of complete dropout or lost synchronization must be addressed. Hamming codes are computationally acceptable and good for errors that are short relative to encoded packet size. Convolution algorithms that depend on continuous reception of long strings of bits may be less robust in the face of complete dropout or long error bursts.

D. OTHER SOURCES

[Blahut 83], [Hamming 86], [Peterson 72] and [Pless 82] are all excellent texts on error-correcting codes. [Proakis 91] provides an advanced-level overview of encoded modulation techniques. [Gray 94] presents an interesting discussion on existing error-encoded modulation techniques and the goal of building a modem which incorporates FEC. An excellent article which focuses on Reed-Solomon codes is [Walker 94]. Some source code is available for experimentation and implementation [Rockliff 91] [Flaherty 92]. The program in [Rockliff 91] is a Reed-Solomon error detection / error

correction routine that is user configurable and comes with a small demonstration script. The GNU ECC.1 program [Flaherty 92] also implements a Reed-Solomon algorithm, which can correct three byte errors in a block of 255 bytes and detect more severe errors.

E. OTHER CONSIDERATIONS

1. Character Strings

At the physical level, the signals carried by most media represents *binary* data because the digital signals in the computer actually represent ones and zeros. All digitized data, (whether video, audio, or text) can be represented by ones and zeros. However, text is very easy to represent in an encoded fashion. Character strings can be used for encoding because any data set can be represented by them. Any file (including binary sound or video files can be encoded in 8-bit ASCII. For example, the Unix operating system supports uuencode, a program which converts a binary file into an ASCII-encoded representation that can be sent using mail. ASCII can also easily be encoded with an error-detection / error-correction routine. It is important to note that 8-bit ASCII rather than the more common 7-bit ASCII must be supported. Although some input devices, such as keyboards, do not support the full 256-character 8-bit ASCII set, 8-bit ASCII is essential if arbitrary binary data is to be represented. [Turner 94] discusses the use of ASCII in designing a language for cooperating AUVs. This allowed operators to assess system state, since messages passed between units were observed on a regular PC monitor, rather than having to record binary communication data and translate it into readable messages.

2. Encryption for Reliability and Security: DES

Military AUV applications sometimes include operations where the AUV is capable of extending the tactical reach of a surface ship or submarine. In situations such as this, security and resistance to interception is vitally important. Although acoustic communications are by their very nature open and easily intercepted by anyone with the right equipment, the data represented by the acoustic signals can easily be encrypted. The

Data Encryption Standard (DES) has been proposed for AUV applications due to the ease of implementation and the level of security provided [Brutzman 92]. Furthermore, the DES Electronic Code Book (ECB) method is ideally suitable for AUV application because it uses short blocks of data that are not chained to a preceding or following data block and does not require frequent confirmation handshaking. FEC algorithms are well-suited to working with encryption algorithms, before or after the encryption takes place. Encryption algorithms are designed to add randomness to a data stream. Many FEC algorithms actually add structure to data stream, which is exactly what encryption algorithms are trying to negate. However, FEC can be added prior to encryption, and encrypted along with the data. This will protect the randomness of the data, and assuming the message is received within the bounds of the decryption algorithm, the FEC algorithm will protect the data inside the encrypted packet until it is decoded.

3. Compression For Efficiency

[Hoag 94], [Walker 94] and [Turner 94] all mention use of compression algorithms to reduce data volume necessary to transfer data between autonomous units. In many ways, these algorithms work hand in hand with FEC algorithms, which sounds ironic in that compression algorithms are intended to compact data whereas FEC algorithms increase redundancy of that same data. However, there is an optimal balance between the two operations which must be reached in order to achieve the best performance from the encoding method as a whole. This is done by possible compression of the data packets prior to reaching the layers following and expansion of the data after reception prior to passing up to the next higher level. FEC coding is effected at the physical layer, thus FEC is operating on data that has been compressed already. It doesn't matter to FEC what the data is, as long as it is able to add redundancy in an optimal method that is better than that gained by compression and no FEC.

F. SUMMARY

Data communications are most vulnerable to errors while in the channel between sender and receiver. Adding data redundancy in the form of forward error correction can help the receiver repair the damaged data, up to limits which are at the discretion of the programmer. Errors in excess of this level will require retransmission. Various encoding methods are available, and most fall into either block or non-block classes. Block codes are implemented using variations of Hamming codes or Reed-Solomon routines. Non-block codes are weaker than block codes, but are capable of being "nested" inside a stronger block code to maximize efficiency and speed. ASCII text is ideally suited for encoding applications, due to its ability to represent various other data forms, such as binary data, sound, or video. Data encryption and data compression are both data-manipulation processes which can benefit from the reliability provided by FEC.

V. TOWARD INTERNET PROTOCOL OVER SEAWATER (IP/SW)

A. INTRODUCTION

Reliability of underwater acoustic communications can be improved by implementing forward error correction coding, but unless higher level protocols are introduced into the acoustic communications channel, all that will have been accomplished is the creation of a point-to-point link. Although that in itself is needed, the IP communications model is based on networks where multiple computers are communicating among each other and communications directed at hosts outside the immediate network are directed out via a specific host, or *gateway*. The function or purpose of Internet Protocol is to move datagrams (packets) through an **interconnected** set of networks [Postel 81]. In this section, reasons for networking underwater acoustic devices are presented, and what work must be done before IP communications can be realized over an acoustic channel. We call the result of this process IP over Seawater (IP/SW). Methods of implementation are presented, including recommendations for initial experimentation with existing serial line point-to-point protocols.

B. NETWORKING APPROACHES TO COMMUNICATIONS

A networked approach divides the detailed functionality of computer communications into well-defined layers. One common model is the four layers of the Internet Protocol (IP) suite [IAB 94] [Stallings 94] described in Chapter II. The IP model is more a loose hierarchy of functionality than a strictly separated set of layers in that interlayer communication is not restricted to adjacent levels. Simplified descriptions of the functional characteristics of the four IP layers appears in Figure II-4, and a representation of the logical content of IP datagrams is shown in Figure V-1. It is interesting to note that the media for commonly understood physical and transport layer network standards have high propagation speeds, high bandwidth capacities and low noise (e.g. Ethernet, FDDI, ATM etc.). This means that the design of corresponding protocols can essentially ignore the

temporal costs of retransmission. Such a situation is not the case in noisy, slow propagation and low bandwidth environments. However, many similar transmission problems have been addressed and solved using a networked approach in other media and environments. In particular, internetworking solutions have been proposed and achieved for wireless environments such as packet radio [Chen 94] and satellite [Jabbari 94]. Investigations into internetworking for seawater will gain much by evaluating the balance struck in these other network domains. An example is wireless LANs.

Wireless LANs are usually set up in either a broadcast (flooding) structure or a structured packet-passing scheme. Broadcast systems simply receive packets, examine them to see if the address is theirs, and if not, they retransmit the packet. In this system, channel utilization is low, but implementation is easy. CSMA, ALOHA, or slotted ALOHA are common examples of this system. Packet passing systems, which elect a spokesman to act as a base station, are network-intensive due simply to maintaining the spokesman. However, most wireless systems have a structured, pre-existing high-speed network that they are tied into, and traffic within a small wireless system can be grouped into two directions: uplink (into the high-speed existing network) or downlink (from the high-speed network into the wireless network). Access points are nodes on the high-speed network which connect to the wireless network, usually base stations or repeaters. More often than not, downlink communications are broadcast from the access point to the entire network on a common channel. This presents a problem if one or more stations did not receive the transmission correctly. If the message was not for them, they would not know that it was not intended for them if the address was the part they did not receive. Therefore, they would NAK and the whole transmission would have to be repeated needlessly. This could happen many times before all stations received the message correctly. An alternative is either to multicast each message multiple times, incrementing a sequence number so that the receivers have to receive the message correctly only once, or else to not require acknowledgements from the wireless nodes. This is an example of an unreliable delivery scheme.

As can be seen, many of the problems which plague the wireless LAN designers are the same as those which underwater acoustic networks must contend with, except the stakes are higher and the failure of a node is not easily rectified as in a wireless mobile LAN.

Acoustic networks must have a medium access control protocol which maximizes throughput, not bit rate. In addition, operating throughput is practically more important than theoretical throughput, a metric which is all too often used to determine network operating health. Requirements for an acoustic network MAC protocol are much the same as those listed for a wireless network in [Chen 94] and are listed in Figure V-2. Of particular note is the segregation of most MAC protocols into three distinct groups: random access such as CSMA, CSMA-CD, or ALOHA, polling, such as described in [Chen 94], [Bharghavan 94] and [Catipovic 93], and time division multiple access. Most work in wireless MAC protocols seems to be focusing on polling, and it is not surprising that acoustic LAN efforts are not directed in the same direction. The ALAN described in [Catipovic 93] requires the access point to resolve multiple transmit request messages from contending nodes, and sort them over time. This places the burden of receiving overlapping and competing transmissions on the access point, when in conventional networks none would be serviced. Each transmitter would simply back off and retransmit and hope for success. This is an example of a protocol which utilizes advancements in hardware to perform layer functions. It will be interesting to see how this system performed.

C. TOWARDS IP OVER SEAWATER (IP/SW)

Extensions to IP have been established over many disparate media. The most fundamental principle for establishing IP compatibility among acoustic telemetry devices is to specify data link and network physical layer functionality. This will consist of physical control, medium access control (MAC) and logical link control (LLC). If interfaces are properly designed, it will be feasible to use acoustic modems and their supporting controllers in the same way as a LAN interface card plugged into a personal computer. Underwater acoustic devices might then be considered replaceable parts. The functionality

of different types of acoustic modems can be interchangeable as long as all the modems utilizing a single acoustic link are acoustically compatible. Although it may not be possible to permit direct control of the physical layer by upper layers, such functionality is optional under IP and is not always present in other network media.

Layer functionality may have to be revised in the light of MAC and LLC protocols, depending on advances in hardware. This idea is represented in Figure V-3. For example, with DSP taking a significant portion of available power, existing layer functionality may have to be modified by reducing or eliminating some or all of a particular layer, depending on what functions that layer contributes. Some of the functions are not needed in an acoustic network, considering the up-link / down-link characteristics of this type of network. Internet address resolution, done at the IP level, will probably be done by the access point using a hash or lookup table and then sending a modified, shortened address to wake up that particular node.

Multiple ALANs might operate simultaneously in close proximity through frequency band separation. The underwater acoustic channel is a shared medium. In this respect transmitted messages can usually be considered broadcast, since all compatible devices will hear a transmission and be required to process it prior to determining packet address. If multiple acoustic channels are used, the acoustic channel is analogous to a multicast channel since individual devices are likely to ignore frequency bands that are not of interest. Given that bandwidth is low and temporal latency high over a single acoustic channel, the use of multicast and multiple frequency approaches will primarily be used to increase effective bandwidth. We estimate that current microprocessors typically have adequate computational capacity to process several acoustic telemetry channels simultaneously. Multiple frequency bands can be particularly useful for a single communications channel. Since blanking occurs when a transducer is transmitting, followed by a short period of time when local reverberation prevents effective reception, an acoustic channel might effectively be considered half-duplex. More efficient acknowledgment/negative acknowledgment (ACK/NAK) schemes use multiple frequency

bands. Given that the source power requirements, propagation paths and attenuation of sound in water are all highly dependent on frequency, and given that one host may be sending much more data than another, it is likely that out of two frequency bands, one is better suited for transmission and will therefore be dedicated to the host that is the predominant transmitter.

One lesson demonstrated repeatedly in network protocol development is that individual protocols in theory may appear efficient, but in practice can incur excessive overhead when combined with other layers of the protocol hierarchy. Current acoustic telemetry devices operate independent of any other network connectivity present at the host computer. It is likely that new trade-offs and optimizations will be necessary when existing acoustic telemetry protocols are adapted for IP/SW compatibility. Packet content may also require redesign, in order to incorporate the functions shown in Figure V-4. In other transmission media, a networking approach to communications architecture development has repeatedly produced overall improvements that are better suited to user requirements than specialized protocols. One additional point is that delivery of individual acoustic IP packets need not always be reliable. The Internet transport layer provides for both reliable (TCP) datagrams and unreliable best-effort (UDP) datagrams. Currently unreliable delivery is the norm in acoustic telemetry devices. Adding reliable delivery without requiring unacceptably expensive retransmission schemes is an essential prerequisite for development of IP/SW. Because all hosts on the internet are not able to connect to a specific IP address on an acoustic network, a gateway is needed which knows the IP addresses of all the acoustic hosts and can strip the IP headers off the packets intended for that host. It can then use a host table or lookup table to find the host's unique address, much in the same way a router finds an Ethernet address.

D. IMPLEMENTATION

Although true acoustic networks will not be realized without MAC layer functionality, one must walk before running. Even with forward error correction, a protocol at the

physical layer for transmitting and receiving data is still necessary. Several protocols already being utilized are the Serial Line Internet Protocol (SLIP) [Romkey 88] or Point-to-Point Protocol (PPP) [Simpson 92] [Simpson 94]. These protocols are utilized by hosts connected to the internet by a serial line, and are used for passing Internet packets through the telephone lines. All SLIP does is define a sequence of characters that frame IP packets on a serial line, and nothing more. It provides no addressing, which is done at the IP level, nor packet type identification or error detection/correction or compression. Because it does so little, though, it is simple and easy to implement. As most serial lines are point-to-point links, it is reasonable that the network protocol proposed for an acoustic network which is intended to utilize SLIP should make maximum use of a point-to-point routing mechanism.

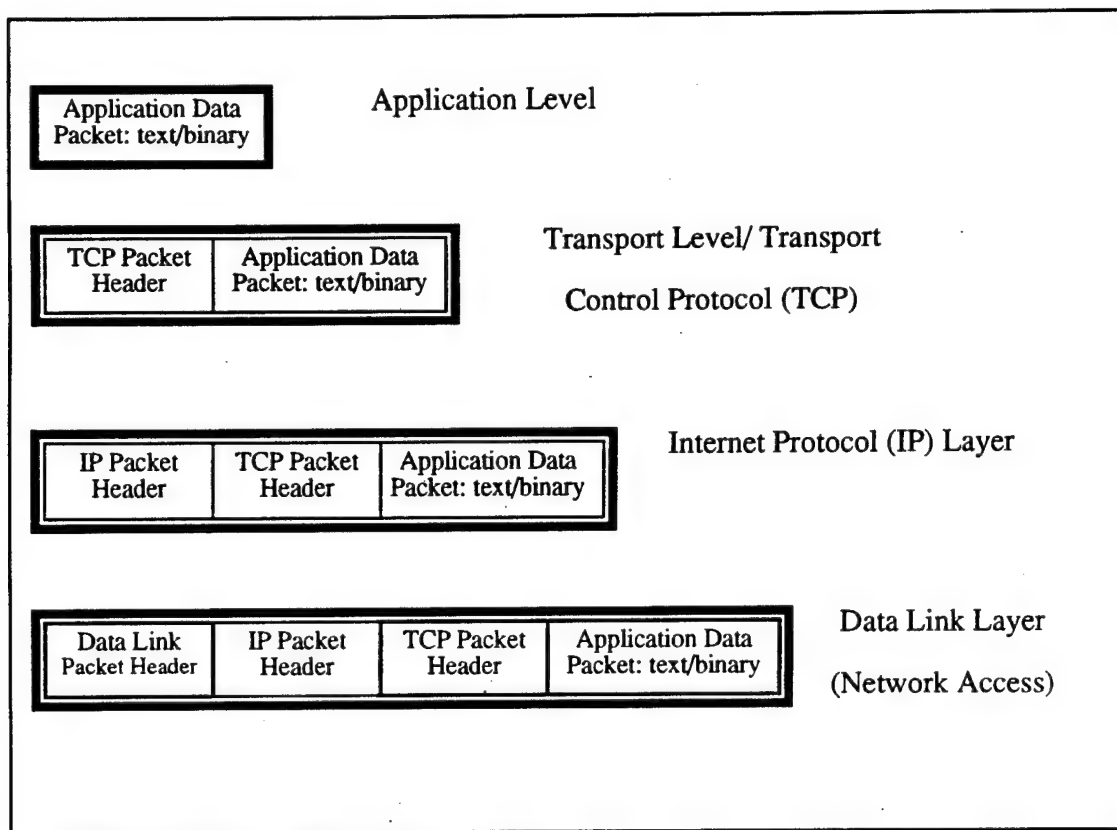


Figure V-1: Logical content of a IP packet.

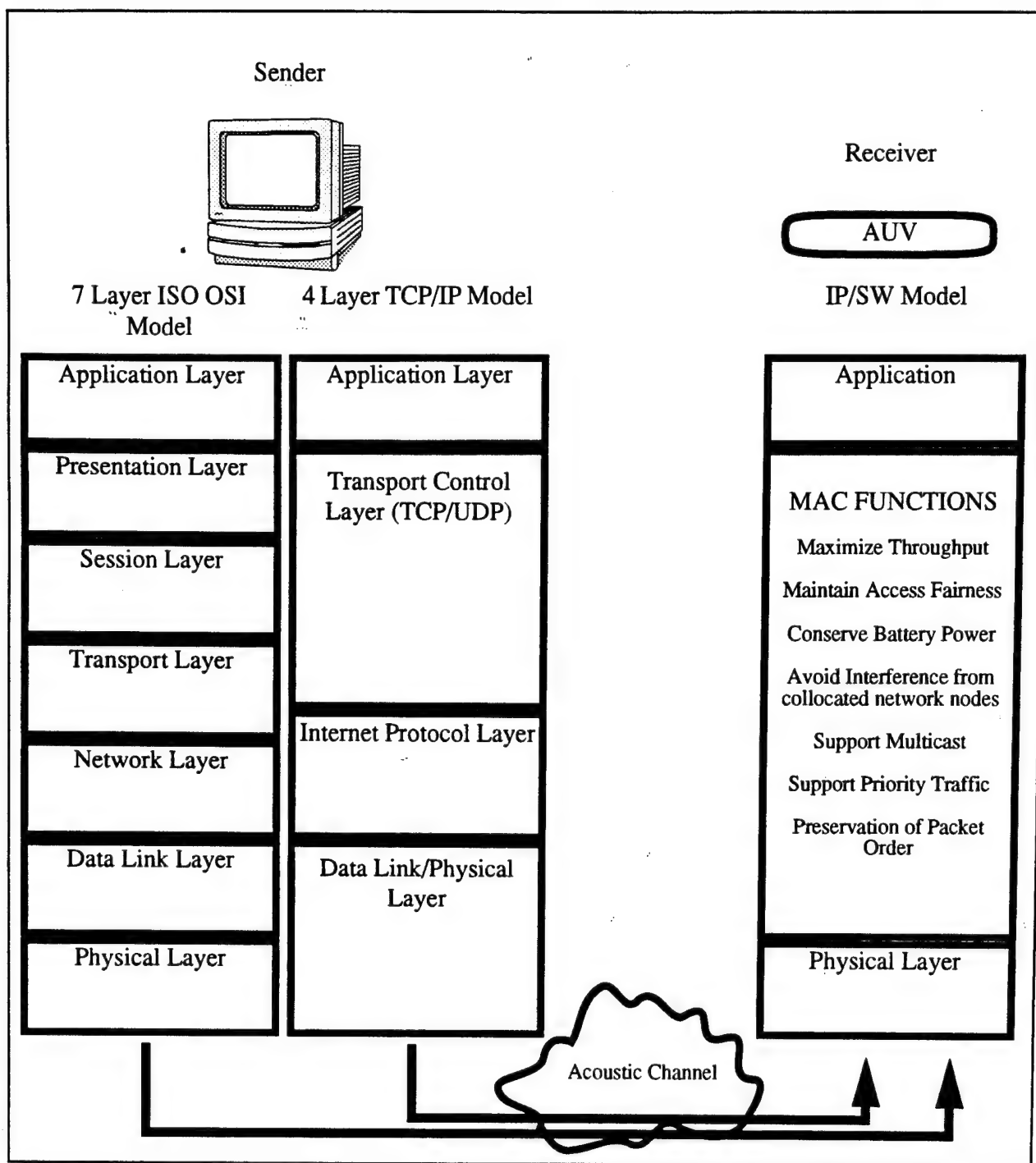


Figure V-2: Logical functional similarities between various network models and MAC level functions required in acoustic LANs.

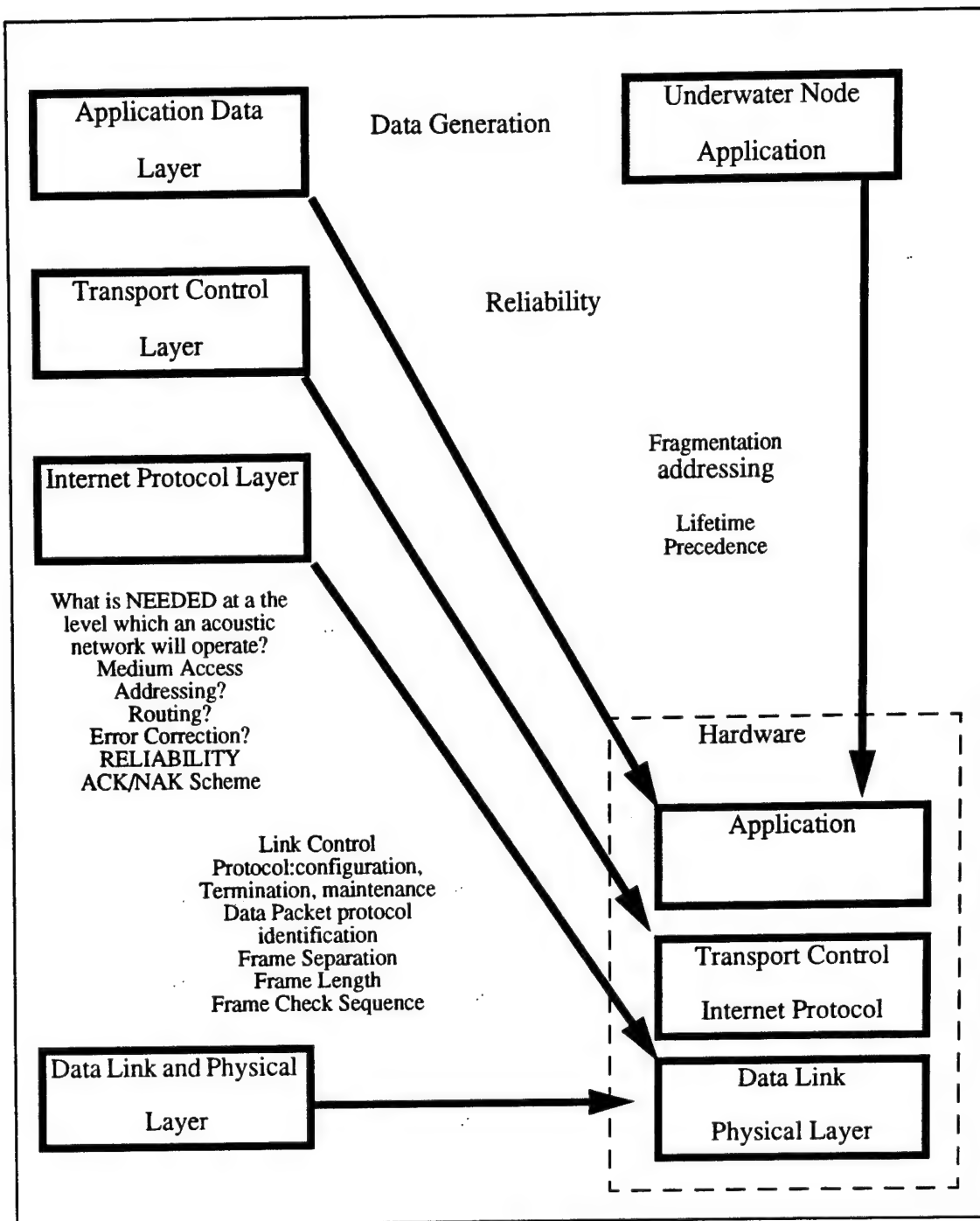


Figure V-3: Physical (Hardware) Level Transmission

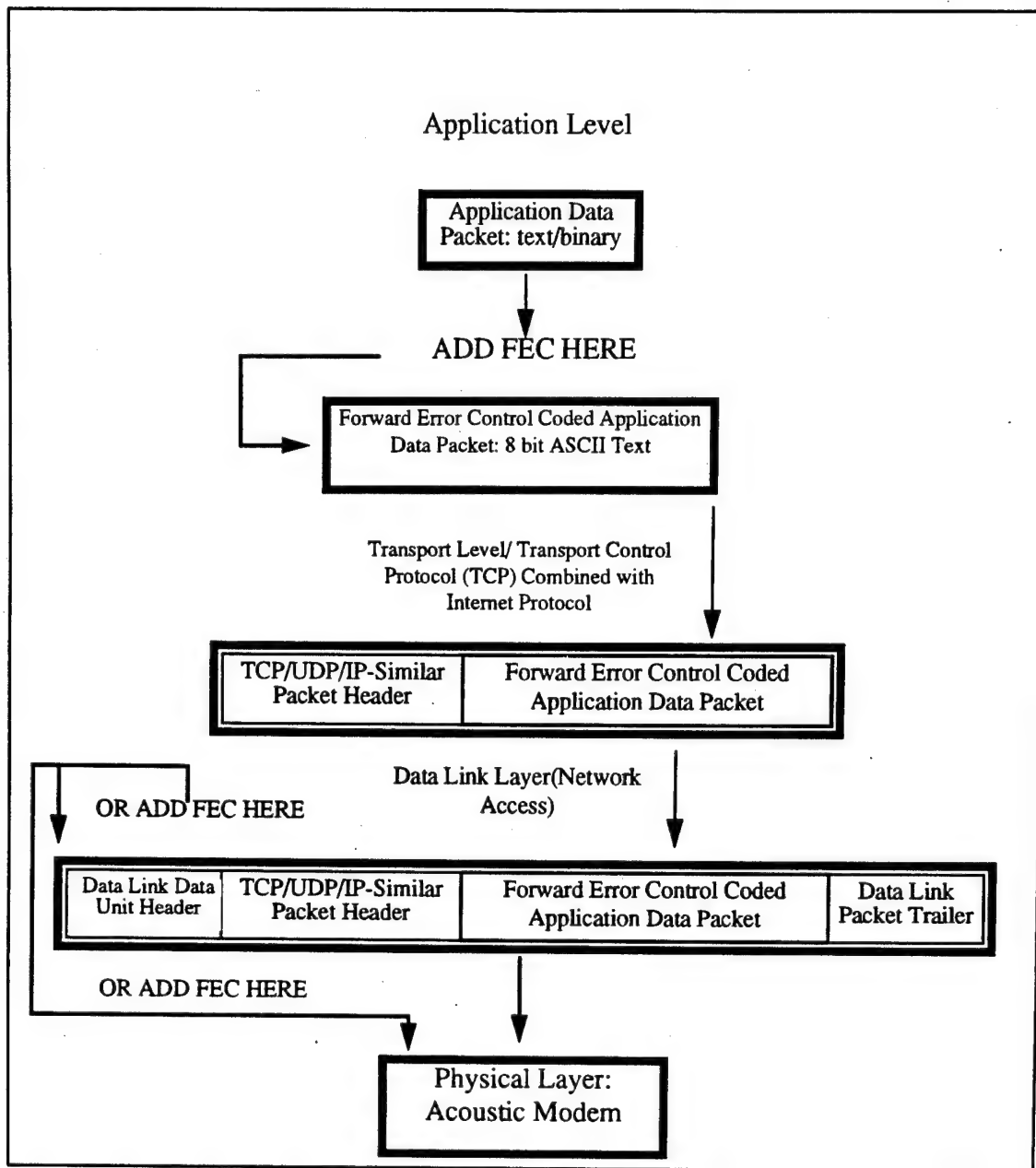


Figure V-4: Logical content of a IP/SW packet.

The Point-To-Point protocol seems to hold promise for initial investigations into point-to-point links, mainly because it requires nothing other than a duplex channel. A

duplex channel is easily acquired with two pairs of modems, described in Chapter II, set to two different frequencies. As long as frequency selectivity is of sufficient quality to avoid cross-channel interference, a point-to-point duplex link is possible. Even with low-cost DiveTracker units running a modem program, this protocol is ideally suited for initial testing.

Determination of an optimum spreading of functionality throughout the IP stack is a viable area of study for IP over Seawater. Some functions should be implemented at every layer, and others need only be used at one. Determination of which function at which layer can only be determined by testing.

E. SUMMARY

As functionality is moved from the upper levels down towards the physical layer, much of what is done in software is able to be implemented in hardware. As algorithms mature, hardware advances parallel the applications and are able to increase speed accordingly. Whether in software and using faster processors, or in hardware and using slower processors, IP over SW is possible either way. The Internet Protocol is flexible and able to be modified to meet the environment it is tailored to operate in. Specific challenges to IP over Seawater will include the medium access problem and retransmissions if contention is encountered. These retransmissions have nothing to do with messages received in error, because they are simply collisions. Hardware capable of resolving such collisions may become available, but other alternatives such as polling are viable options and should be subjected to further testing. Optimization of layer functionality will result in a trimmed-down, efficient protocol structure which embeds redundancy where it is needed and eliminates excess baggage where not needed. Results of this optimization can be implemented in a modified packet structure, as long as standards organizations can agree on what is required and what is optional.

VI. EXPERIMENTAL SETUP AND RESULTS

A. INTRODUCTION

No research is complete without experimentation, and research in acoustic communications is no different. This section describes the work completed in proving the efficacy of forward error correction to reduce the need for retransmissions of a message received in error. First, a computer-to-computer simulation is described, which implemented a error-injection routine into a stream serial data. Second, results of in-water tests both in the laboratory and in the North harbor of Moss Landing using DiveTracker diver communication devices as modems are reported.

B. SETUP

The experimental goal of this thesis is to show the effectiveness of a 12/8 Hamming error-correcting code in comparison to transmission without forward error correction. The data was obtained in two separate steps. The first step was to create a Hamming encoder/decoder and run it on two personal computers to demonstrate the effectiveness of forward error correction. Acoustic noise was simulated by assuming that no more than one bit in 8 (or one bit in 16) was affected by error and was inverted. This was done to simulate a 10% data error rate, a fairly typical value for an underwater system operating at the boundaries of usefulness [Flagg 95]. Long series of errors corresponding to acoustic transients such as overhead shipping are a special case and were not specifically evaluated. Different FEC codes can be applied to ameliorate long-duration error bursts and are an excellent topic for future experimental work.

In preparation for in-water testing using DiveTracker acoustic modems [Flagg 95], data encoding software was adapted for end-to-end testing using two PCs connected by null modem cables via serial ports. The test equipment and data flow diagram appears in Figure VI-1. The only difference between the in-water rig and the simulation rig is absence of the DiveTracker sonar devices and the insertion of synthetic noise during simulation.

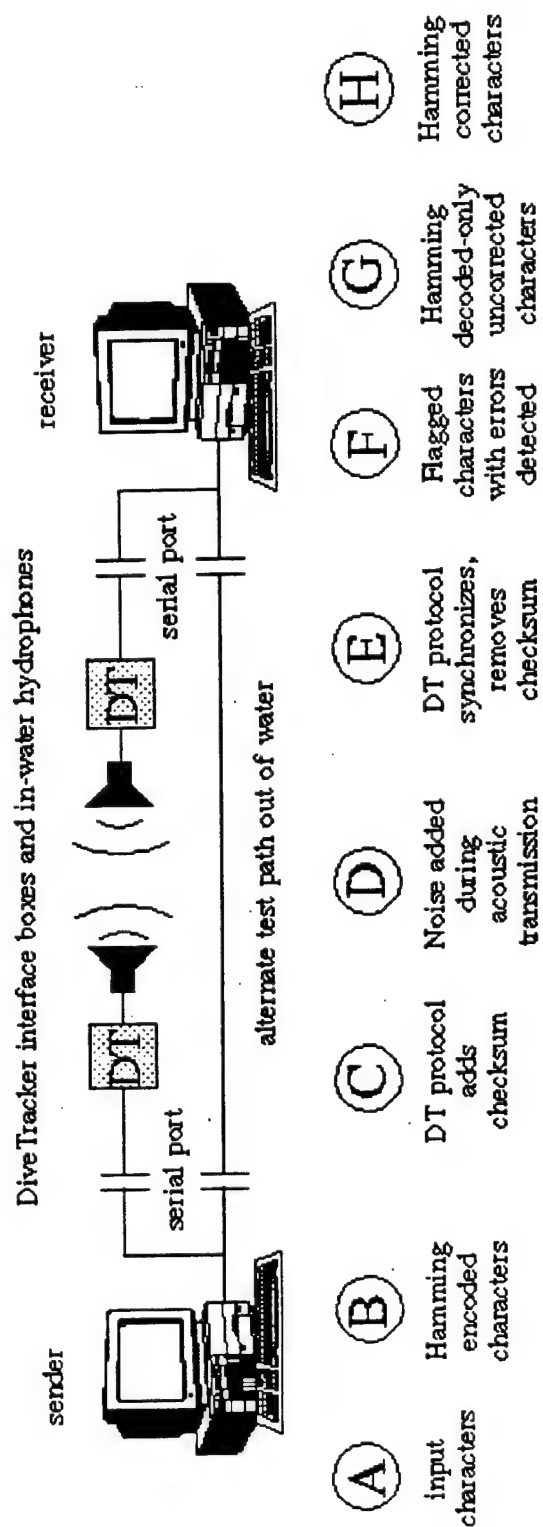
An initial test scenario called for 100 repetitions of a fifty character message, "*The purpose of computing is insight, not numbers!!*" [Hamming 86]. Transmission of a fixed character string parallels the testing of modems described in [Garmer 92]. Measures of effectiveness included correct operation of the Hamming encoder/decoder routines in the presence of single bit errors, percentages of garbled messages corrected at the receiver, (H/A), and percentage of garbled messages uncorrectable at the receiver which would require retransmission to receive correctly. Simulation term definitions appear in Table VI-1. The second test was to actually use an acoustic modem to duplicate this scenario while passing data through water and observe the effect of the error-correction code on data transmission.

C. HAMMING ENCODER/DECODER SIMULATION

The experiment used Hamming 12/8 double-error detecting/single-error correcting code with even parity, which means 12 bits total were used to encode 8 bits of message using 4 bits for parity. These values were selected because 8 bits corresponds to a single byte and a byte-base coding algorithm was desired for the reasons discussed in Chapter IV. Double-error detecting indicates that a single bit or double bit error out of the 12 total bits can be detected. Three bits or more in error out of 12 are likely to produce an internally consistent result which cannot be detected by this code. Single-error correcting implies that any single bit in error out of the 12 can be corrected using the redundant information present in the parity bits. Figure VI-2 shows how message bits and parity bits are combined to convert two uncoded message bytes into three Hamming-encoded message bytes. Thus two uncoded 8-bit ASCII characters are represented by three encoded 8-Bit ASCII characters. Even parity implies that any valid combination of message bits and parity bits must add (modulo 2) to zero. The actual code was derived from [Hamming 86] and is listed in Appendix A. The Hamming encoder/decoder routines proved correct for all cases in a wide variety of test strings, some of which included special 8-bit ASCII characters to show general applicability to arbitrary binary data. Simulation results in appear in Table VI-2.

Data Point	Description	Character Counts, Percentages	Comments
A	Original string sent by user		"The purpose of computing is insight, not numbers!!
B	Total bytes for Hamming encoded characters		Parity bits added, 12/8 code increases total size by 50%.
C	DiveTracker pulse-position coding adds synchronization to two bytes + checksum, if on	Simulation tests replace C/D/E with 1 in 8 or 1 in 16 random position bit error	Synchronization of pulse is essential.
D	Noise added during acoustic transmission, propagation, and reception		Empirical noise distributions will be far more complex.
E	DiveTracker pulse-position coding delivers two bytes		DiveTracker checksum error causing retransmit will be disabled during in-water tests.
	1 out of 8 or 1 out of 16 bits eligible for error		Arbitrary noise probability function. Position of error bit in byte is also random.
	Probability that a single bit is in error		
A - E	Errors induced (in water or simulation)		One or more bits/byte, unknown by receiver.
F	Flags marking characters with errors detected		Detects single, double, and multiple bit errors.
G	Hamming (decoded-only) uncorrected characters		Original sentence restored, transmission errors still present
H	Hamming corrected characters		Assumes only single error corrected
	Undetected / uncorrected errors		Unknown by receiver.
$\frac{A-E}{A}$	% received characters in error prior to applying correction		Expected retransmission needed if FEC not used.
	% characters correct and corrected		Overall FEC robustness.

Table VI-1: Explanation of simulation parameters.



(1 bit in 8) or (1 bit in 16)
random noise generator used in simulation

Figure VI-1: Test setup with letter labels corresponding to data sampling

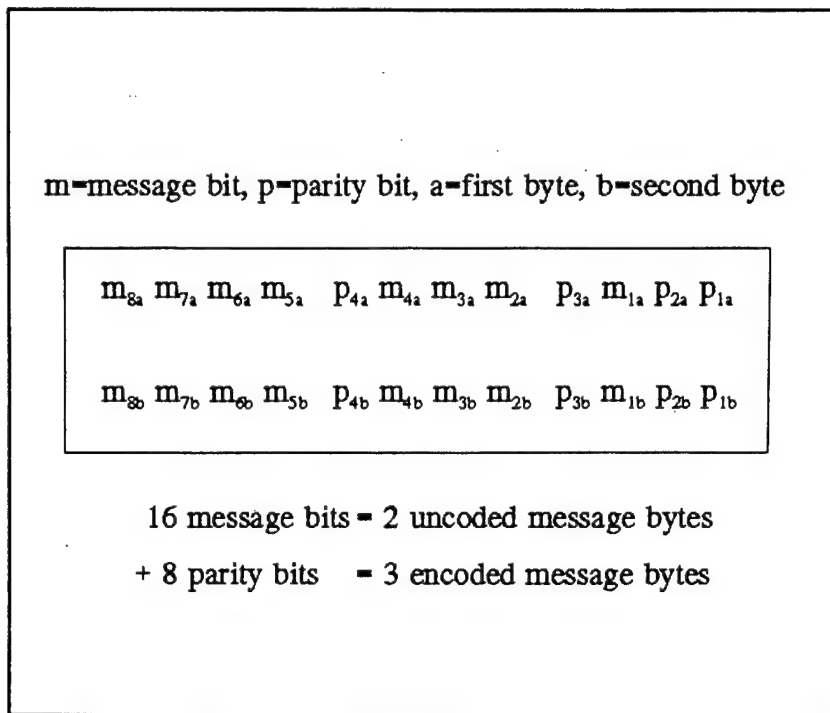


Figure VI-2: 12/8 Hamming code structure.

Generation of the encoder and decoder was done in the C programming language. This language was used mainly because of the large amount of existing code available which was needed to implement the encoder-decoder on a PC using serial communications. This is also compatible with DiveTracker, NPS AUV, and other underwater robot software environments. The C programming language was learned outside of class using [Aitken 94]. It was found to be nearly impossible to learn a language such as this without the help of someone familiar with the language to clarify points of confusion, of which there were many. Pointers were difficult to learn and use properly. It was found that an understanding of the concepts of data representation in C is critical for effective use of bitwise operators and type conversions. A Borland C++ compiler was installed on a PC for use in compiling the code for the encoder/decoder, since it was to run on a PC.

The program evolved by first using a main function which passed a string of text to an encoder function. The encoder produced a string 1.5 times larger than the input string, due to the additional Hamming parity bits embedded in each of the encoded characters. This array was then passed to an error simulation function, which injected random errors into the array of text to be transmitted. The encoded text, now containing errors, was passed to a decoder function, which read it in and corrected the errors as they were encountered on a character-by-character basis. The input string, the transmitted string, the erroneous-received string, and finally, the corrected received string were displayed on the video monitor. The code was then separated into two different functions for respective operation on a transmitter and a receiver. This test setup is shown in Figure VI-3.

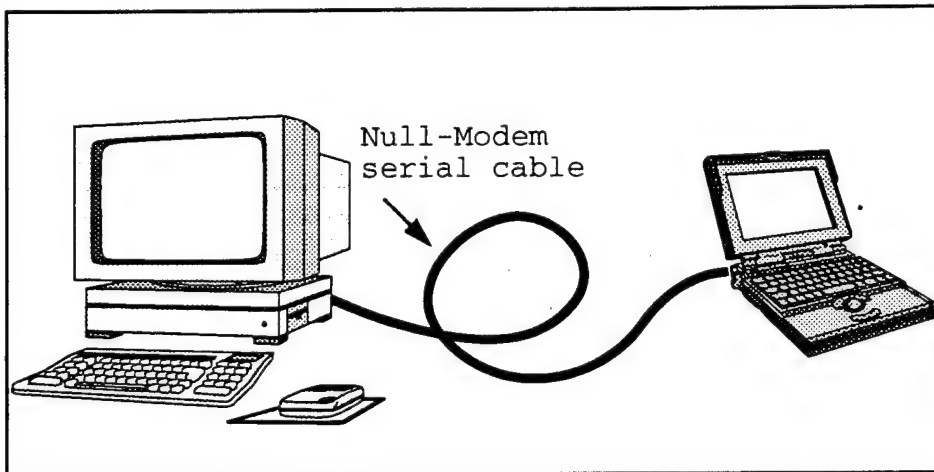


Figure VI-3: Initial test setup for error simulation.

D. COMPUTER SETUP

A null-modem serial cable was attached between the two computer's serial communication ports. Null modem cables are required whenever two DTEs are required to communicate between each other because of the way their communications ports are designed. Each DCE is designed to be paired with a DTE, and their mechanical and electrical characteristics are such that a standard serial cord will suffice to do the job. However, when two DTEs are joined using a DTE-DCE cord, the transmit line of one DTE

will be attempting to communicate with the other's transmit line instead of its receive line.

Data Point	Description	Character counts, percentages				
A	Original string sent by user:	5000	4000	5000	5000	2500
B	Total bytes for Hamming encoded characters	7500	6000	7500	7500	3750
C	DiveTracker pulse-position coding adds synchronization to two bytes + checksum, if on					
D	Noise added during acoustic transmission, propagation, and reception					
E	DiveTracker pulse-position coding delivers two bytes					
	1 out of 8 or 1 out of 16 bits eligible for error	1/8	1/8	1/16	1/16	1/16
	Probability that a single bit is in error	0.1	0.2	0.1	0.4	0.8
A - E	Errors induced (in water or simulation)	752	1161	374	1537	1500
F	Flags marking characters with errors detected	728	1084	374	1537	1500
G	Hamming (decoded-only) uncorrected characters	5000	4000	5000	5000	2500
H	Hamming corrected characters	728	1084	374	1537	1500
	Undetected / uncorrected errors	24	77	0	0	0
$\frac{A-E}{A}$	% received characters in error prior to applying correction	15.0%	27.1%	7.5%	30.7%	30%
	% characters correct and corrected	99.5%	98.1%	100%	100%	100%

Table VI-2: Simulation Results

Hence, a device to allow the transmit line of one computer to communicate with the receive line of the other is needed. This device is called a "Null-Modem." A null-modem cable

differs from a simple extension cable in the way that the two ends are connected in order to allow two computers to communicate with each other using an RS-232 interface. In addition to connecting the two computers correctly, there are a couple of additional steps that must be completed before the computers can communicate. One of these involves configuration of Interrupt Requests (IRQ). Interrupts are addresses where the CPU looks to see if there are any waiting tasks that must be serviced. In setting up a serial communications program, the serial port must be configured with the proper IRQ number. Complete details on this approach can be found in numerous references including [Campbell 87].

E. ACOUSTIC COMMUNICATION WITH DIVETRACKER: *AMODEM*

DiveTracker diver navigation and communication devices described in Figure II-7 and Figure II-8 were used as acoustic modems. The DiveTrackers were loaded with a program called *AMODEM.DC* which takes two bytes at a time from an RS-232 serial line and transmits them in a 20 bit word using MFSK. *AMODEM* is user configurable for the parameters listed in Table VI-3. Appendix B explains how the code is loaded into the two DiveTrackers and executed. The receiving units then export the two received bytes out to

Speed	Power	Pulse Length	Gain	Threshold	Filter	Checksum
0-4 (0 = 25 bps) (1 = 64 bps) (2 = 125 bps) (3 = 250 bps)	000-255 (% of duty cycle of the driver waveform) (0 = 0%) (255=50%)	0100-5000 (microseconds)	0-3 (10 dB over previous step)	0-99 (Every 10 decreases sensitivity by 6 dB)	0-5 Filter ID	0/1 Off/On
1	255	1000	3	16	0	1

Table VI-3: *AMODEM* default power-on parameters [Flagg 95].

their respective personal computers running a terminal program. The Hamming code which had been installed on the PCs for the previous test is then executed. It is important to note that the speed on both modems must be set the same, or communications will be garbled.

In all the tests run with the Hamming code, the Checksum feature was turned OFF. The default power-on settings of *AMODEM* were left alone, with the exception of the power settings which were reduced to 10 for most of the tests.

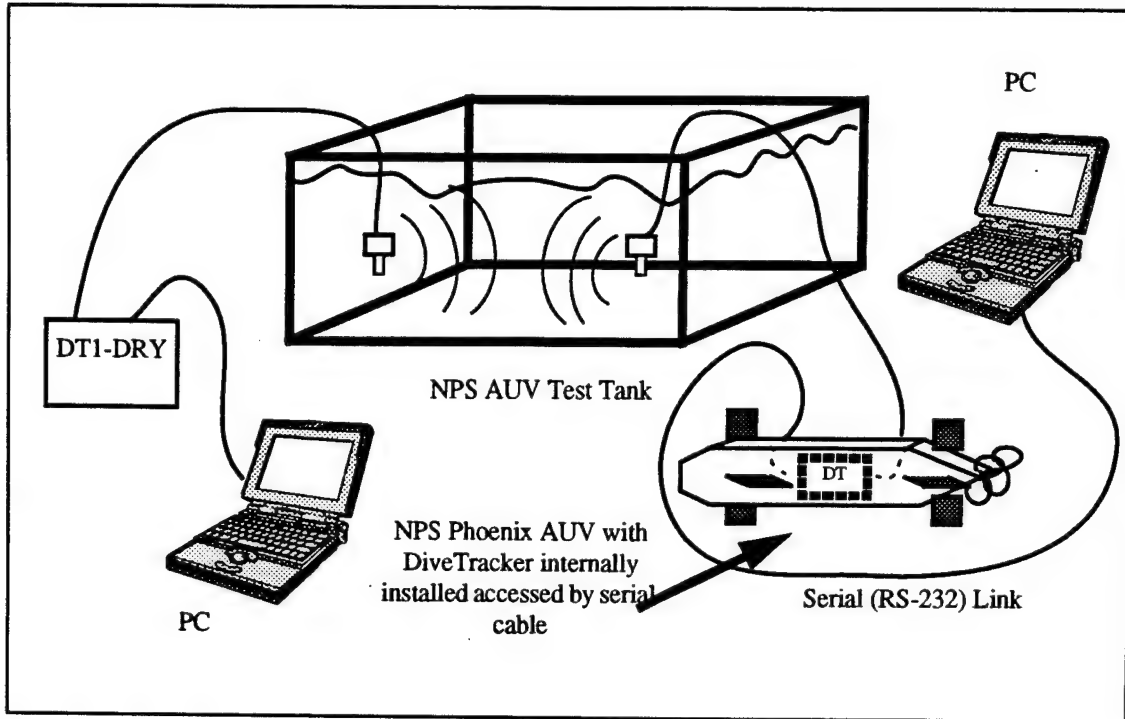


Figure VI-4: DiveTracker Setup in AUV Test Tank

F. AUV TEST TANK

The first tests of the Hamming encoder/decoder using the DiveTracker modems was done in the NPS AUV laboratory test tank as shown in Figure VI-4. This tank is an above-ground structure made out of steel, 20 feet long and 20 feet wide, 8 feet in depth. The transducers were initially hung in the tank with about 6" separation between them. The terminal program was executed, but no data was displayed by the receiving computer. Various power levels, pulse lengths, and threshold parameters were all attempted, but no connection was made. To verify correct operation of the system, the transducers were taken

out of the tank and put into a bucket of water. Data was transmitted and received in the bucket without any problems, leading to the determination that the steel tank had too much reverberation and ringing for effective acoustic communications.

Previous experiments with the DiveTracker being used for AUV location and navigation in the test tank had also met with limited success, due to the extreme interference experienced from reflected waves. Acoustic telemetry in a tank constructed such as this was found to be nearly impossible, due to the reverberations and echoes that occur off the steel walls of the tank. Further test showed that communications were possible when using greatly reduced power levels and longer blanking intervals.

G. THE TEST PIPE

After finding that the test tank would not support data communications, a different approach was taken. Due to the difficulty of moving the AUV, which had the DT1-Mod DiveTracker module enclosed within it, to a suitable test location, a test pipe was set up to create an acoustic channel. The first attempt used 35 feet of 2 inch diameter PVC pipe shaped like a U with 90 degree elbows on each end, all filled with water. Unfortunately, data could not be passed in this pipe at all. The next approach used a 35 foot-long section of 4" PVC pipe, with 90 degree elbows on each end and filled with water. This setup worked well and is shown in Figure VI-5.

H. PROBLEMS WITH DIVETRACKER

A problem with the DiveTracker was discovered in that data errors would show up in the receiver when the Hamming transmitter/receiver code was first implemented. The errors were consistent in that they occurred at the same place in the string of received data, and they displayed the same characteristics. The problem was finally discovered to be lack of flow control from the PC to the DiveTracker, causing a buffer in the DiveTracker to fill up and overflow. After confirming this with Desert Star Systems, it was determined that the buffer capacity was 256 bytes. To correct this, handshaking between the PC and the DiveTracker needs to be implemented. Better yet, *AMODEM* should be modified to send

an X-OFF when the buffer is more than 3/4 full and send an X-ON when the buffer has been emptied to 1/2 capacity. For the test this problem was corrected simply by inserting a forced wait routine in the character sending process, which effectively gave DiveTracker a chance to stay ahead of the data flow. However, this situation shed light on the whole concept of serial communications and flow control. It is recommended that more RS-232 compliance be built into *AMODEM*. Request To Send (RTS) and Clear To Send (CTS) functions need to be built into Divetracker code.

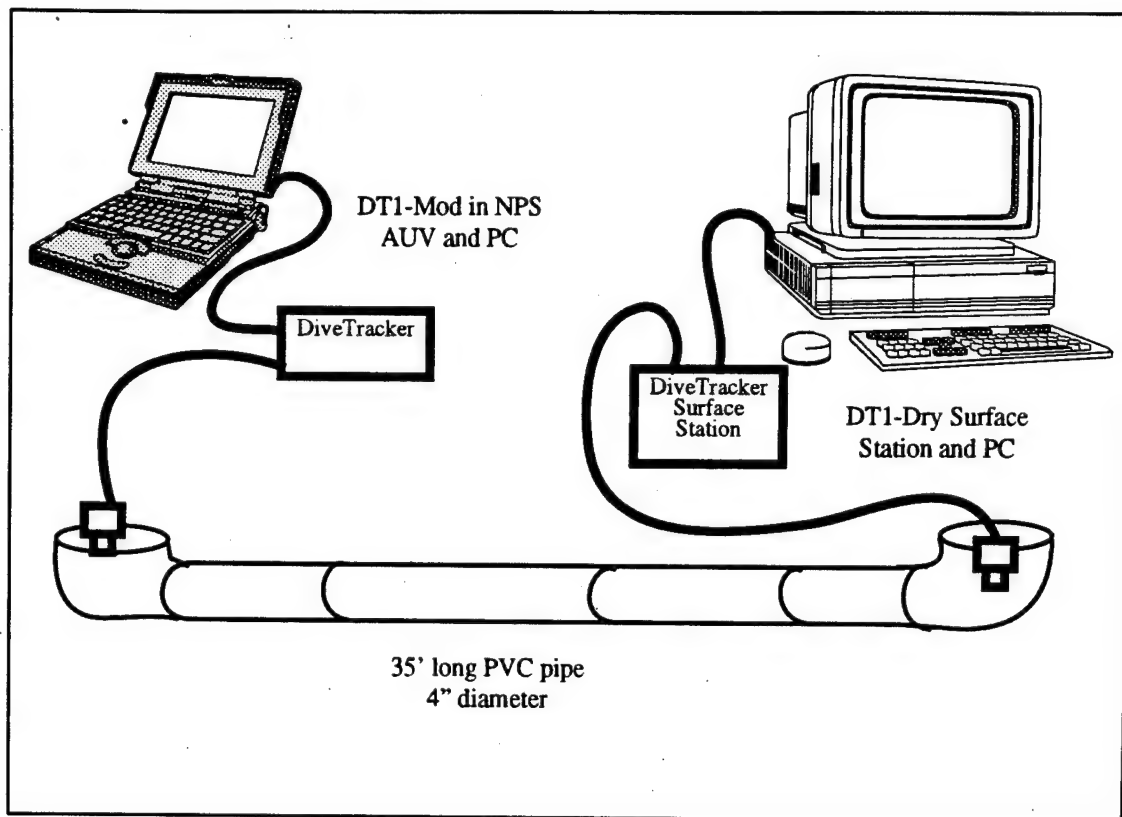


Figure VI-5: Data Communications in PVC pipe.

This is a prime example of the requirement to re-think every aspect of re-designing a protocol foundation. Even the simple things like flow control between communicating hosts needs to be looked at again in view of the application the protocol is being designed

for. You must be prepared to test and troubleshoot at all layers of abstraction in order to validate correct operation.

A problem with the AUV module DiveTracker was also experienced in that for some unknown reason, occasionally the flash memory which stores *AMODEM* and *SMARTDIVE* became corrupted. This problem would reveal itself when the AUV module was powered up and *DIVETERM* executed. The memory contents had extended ASCII characters in the menu displayed by *DIVETERM*. This problem was resolved simply by erasing the flash memory and reloading the programs. This problem repeated itself several times, all in the AUV DiveTracker module.

I. MOSS LANDING TEST PLAN

In order to validate the Hamming encoder/decoder system in an actual environment suitable for AUV operations, the North harbor of Moss Landing was selected for a test location. Factors influencing this decision were the location's lack of boat and tourist traffic, ease of access, and a dock which eliminated the need for a boat. Figure VI-6 shows a gross view of Moss Landing proper, and the close-up in Figure VI-7 shows the actual transducer locations for the tests. Initial tests were conducted with the diver's station and a portable notebook computer as the transmitter, and a surface station connected to a laptop computer as the receiver. In all tests, the receiver was located at the finger pier Point "A" in Figure VI-7 and transducer depth was about 6 feet. The first tests were conducted between Point "A" and a point on the same finger pier about 20 feet away and involved simple string transmissions without Hamming encoding to test the system. During these tests, the maximum speed possible was determined to be "3", corresponding to 250 Baud. Tests 1-5 were conducted to verify the correct operation of the system and are presented in

Table VI-4. The transmitter set was then moved to Point "B" and tests 7-9 were sent using

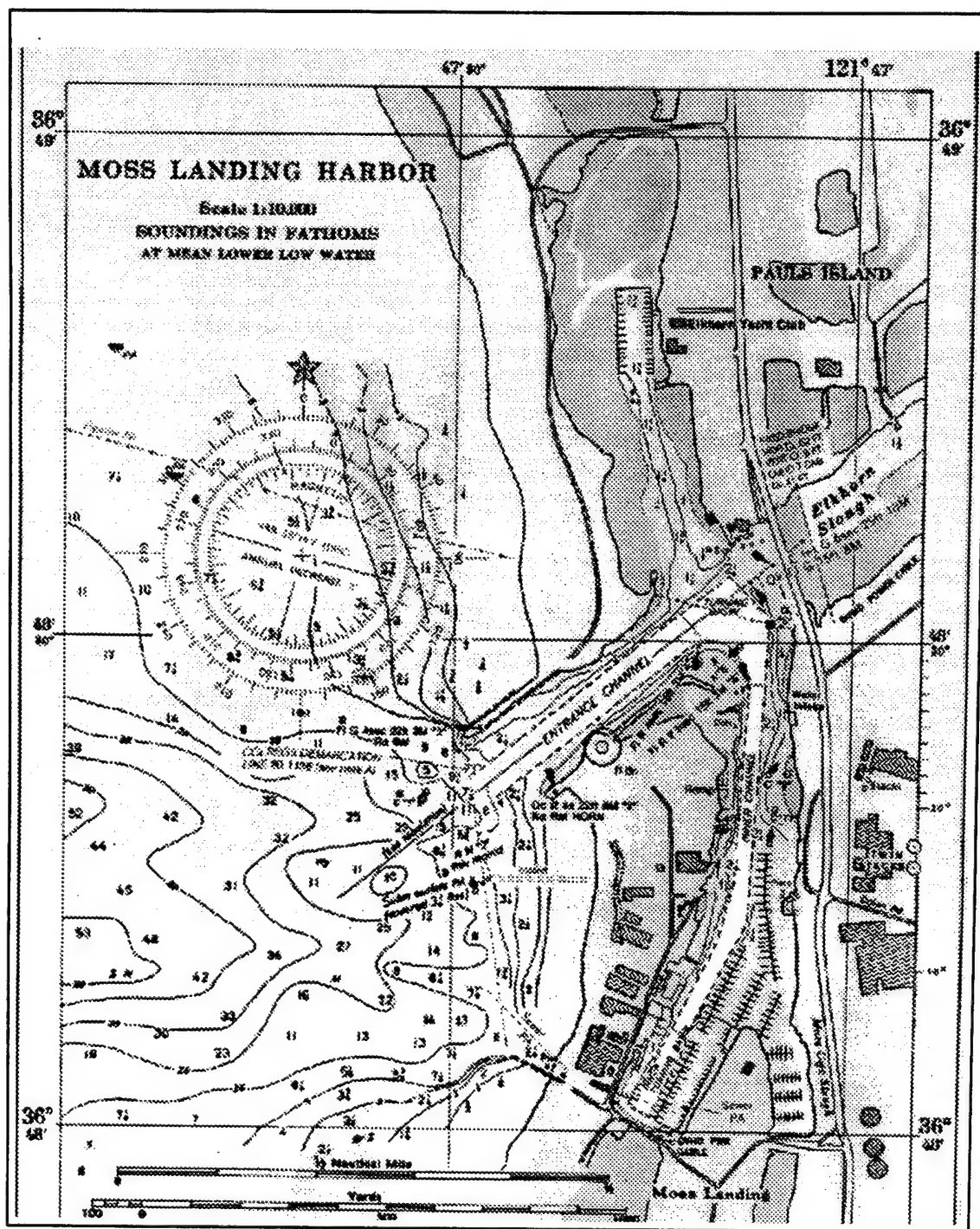


Figure VI-6: Moss Landing Harbor, Moss Landing, California
[NOS 92]

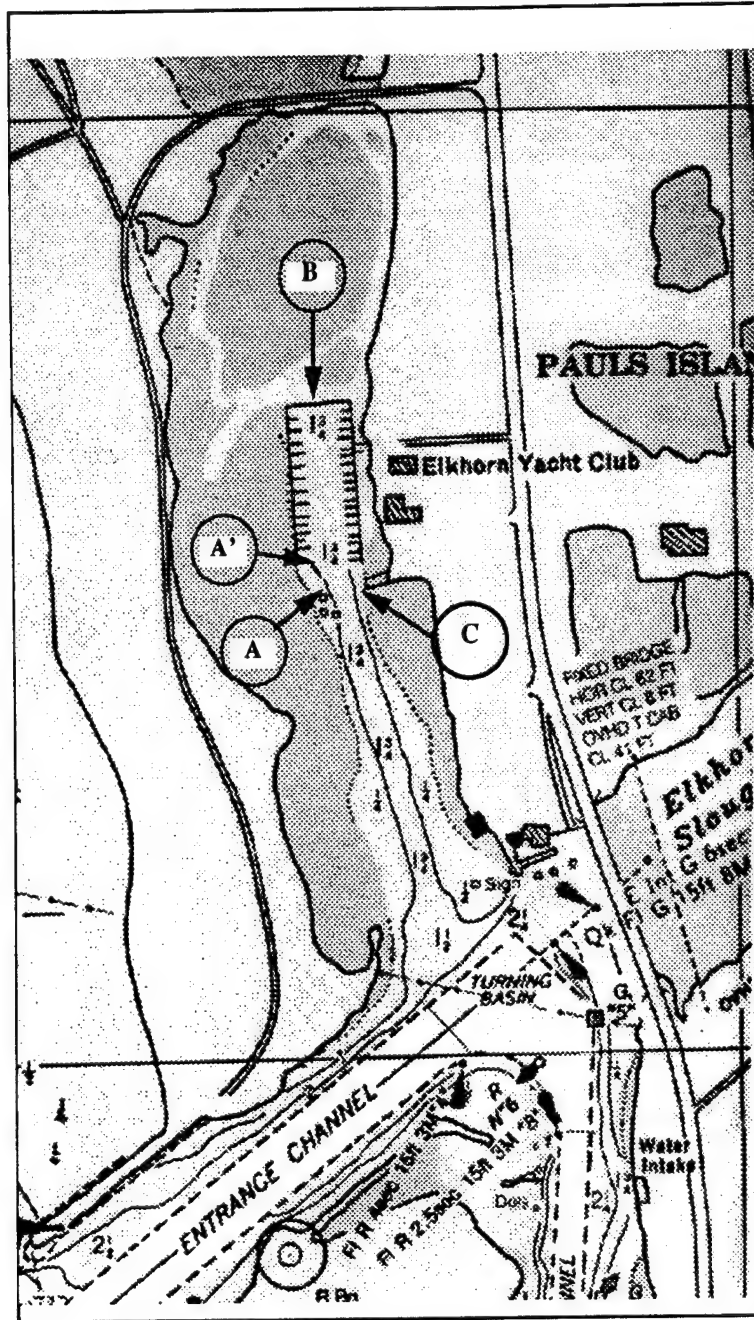


Figure VI-7: Transducer positions for Moss Landing test [NOS 92].

Test #	# of 50 byte strings	Speed	Power Level	Pulse Length	From	To	Remarks	Test File Name
1	20	2	255	1000	A'	A	Uncoded, no errors	1.tst
2	5	3	255	1000	A'	A	Uncoded, one error	2.tst
3	10	4	255	1000	A'	A	Uncoded, many errors	3.tst
4	10	3	255	1000	A'	A	Uncoded, no errors	4.tst
5	5	3	255	1000	A'	A	Coded, no errors	5H.tst
6	5	4	255	1000	A'	A	Coded, many errors, data unusable	6H.tst
7	5	2	255	2500	B	A	Uncoded, many errors, data unusable	7.tst
8	20	2	255	5000	B	A	Uncoded, many errors	8.tst
9	5	1	255	5000	B	A	Uncoded, one error	9.tst
10	5	2	255	1000	C	A	Uncoded, no errors	10.tst
11	5	2	128	1000	C	A	Uncoded, less power	11.tst
12	5	2	064	1000	C	A	Uncoded, some errors	12.tst
13	5	2	064	1000	C	A	Coded, 4 errors, all corrected	13H.tst
14	5	2	064	1000	C	A	Uncoded, no errors	14.tst
15	5	2	064	1000	C	A	Dropped synch, many errors	15H.tst
16	5	2	064	1000	C	A	Received errors, fixed some	16H.tst
17	5	2	064	1000	C	A	Uncoded, some errors	17.tst
18	5	2	064	1000	C	A	No errors	18H.tst
19	5	2	064	1000	C	A	No errors	19H.tst
20	5	2	048	1000	C	A	No errors	20H.tst

Table VI-4: In-water test data.

Test #	# of 50 byte strings	Speed	Power Level	Pulse Length	From	To	Remarks	Test File Name
21	5	2	024	1000	C	A	No errors	21H.tst
22	5	2	12	1000	C	A	Unreadable	22H.tst
23	5	2	18	1000	C	A	Dropped synch in 4th string	23H.tst

Table VI-4: In-water test data.

a longer pulse length. Specific parameters for each test are displayed in Table VI-4. Data received was badly garbled. Pulse length was lengthened again, but the data was still garbled. Finally, transmission speed was reduced until data was received without error at Point "A." As communications between test teams was done verbally, it was decided to move to Point "C," in order to get data with some errors in the transmitted string without dropping synchronization and see how the Hamming code would repair the errors. At Point "C," power levels were dropped until errors were seen, in test number 13. Additional tests were run, but the results did not change as power continued to be reduced. Finally, at power level 12, complete loss of data was experienced. Upon raising power to 18, no errors were seen. Unfortunately, 2 tests out of 23 yielded data that was usable. However, these two tests are significant in that ERRORS WERE CORRECTED. This data will be used in the same table as was the simulation data.

J. DATA RESULTS

Results of the simulation and the in-water tests are displayed below in Figure VI-4 . Table VI-2 is the results from the PC-PC simulation.

1. DISCUSSION

Analysis of the data from Tests 13 and 16 shows that in test 13, 4 errors were received, 4 errors were detected, and 3 were correctly corrected. The one that was incorrectly corrected would have nonetheless signalled an operator that this message would probably

require a retransmission. However, 3 retransmissions were avoided. In Test 16, 9 errors were seen received, 6 were correctly corrected, 3 were incorrectly corrected, and 3 were generated in the output string in error.

Data Point	Description	Character counts, percentages	
A	Original string sent by user:	250	250
B	Total bytes for Hamming encoded characters	375	375
C	DiveTracker pulse-position coding adds synchronization to two bytes		
D	Noise added during acoustic transmission, propagation, and reception		
E	DiveTracker pulse-position coding delivers two bytes, with errors		
	All bits eligible for error	1/8	1/8
	Probability that a single bit is in error	Unknown	Unknown
A - E	Errors induced (in water or simulation)	4	9
F	Flags marking characters with errors detected	4	14
G	Hamming (decoded-only) uncorrected characters	250	250
H	Hamming corrected characters	3	6
	Undetected / uncorrected errors	1	3
$\frac{A-E}{A}$	% received characters in error prior to applying correction	4 of 250 = 0.016. %	9 of 250 = 0.036%
	% characters correct and corrected	247/250 = 98.8%	244/250 = 97.6%

Table VI-5: In-water results for Test #13 and #16.

It is probable that this is due to some of the actual Hamming parity bits being damaged and the correction algorithm making a best guess as to how to fix it. At any rate, errors were detected and the operator signalled. In this test, 3 retransmissions were also avoided.

This experiment shows how interrelated power, speed, pulse length, water depth, and numerous other factors all work together to form a specific set of parameters for each communications channel. It is significant that data transfer was possible after REDUCING power output and pulse length. Perhaps reverberation, multipath, or interference from berthed vessels had a destructive effect on the acoustic communications. Whatever the cause, it presents another challenge: to devise a optimum parameter determination program that each end of a communications channel can run to find the precise combination of parameters for a particular location and time. One possible way to do this would be to synchronize both ends of the link by system clocks, then run a time-based testing program that would record transmit and receive results based on an actual real-time clock. In order to determine what state each computer was in during each test phase, it would only require examination of the data log with the clock entry.

The bottom line of this section is that although not a great deal of data was usable, the idea that reliability of transmitted data can be improved was shown successfully.

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Although many different acoustic communications devices exist, little or no compatibility exists among them. Research efforts in acoustic communications have focused on increasing bandwidth at low bit-error (BER) rates without emphasizing reliable message delivery. Examination of the Internet Protocol (IP) protocol suite shows that IP/SW is a possible network protocol which can help make acoustic telemetry components interchangeable and efficiently structured.

The goal of IP/SW is to provide direct connectivity between underwater acoustic devices and the global Internet. Reliable delivery of communications packets is a critical capability missing from current acoustic telemetry devices. Using example Hamming error-detection/error-correction codes, simulation and in-water testing shows how FEC can improve message reliability while reducing prohibitively expensive retransmission requests.

Any level of correction is possible, but environmental noise be characterized and exploited by coding algorithms. For example, the noise created by a powerboat moving at high speed nearby creates a completely different type of noise than a piledriver or snapping shrimp. Specific additional conclusions follow.

1. FEC, not ARQ

Forward error correction can reduce the number of retransmissions required by 3 to 15 percent, based on experimental results. Although these figures reflect the simulation data instead of the in-water test data, it is believed that with sufficient tuning of the many competing parameters in the acoustic modems, these figures can be realized during further testing.

2. Testing is Paramount

Valid data can only be obtained by doing actual tests. Difficulties in finding a location, communication with assistants, sources of power, weather, and accessibility are all reasons for not doing actual in-water tests. However, as can be seen in this thesis, simulations do not tell the whole story. There was no way to duplicate the conditions or errors received when transmitting in real water during the simulation previously run. Moreover, actual testing provides insights into interrelated parameters and conditions which open more questions for further study.

3. FEC is Easily Implemented

Although these tests used a basic Hamming code, there are other possible codes that are slightly more involved. Appendix E lists two Reed-Solomon encoder/decoders which were not able to be implemented due to time constraints. Although slower than hardware, software is quickly prototyped and much cheaper to experiment with than is hardware. We expect that Hamming codes are close to an optimal general solution. Extensive testing may show other codes to be more efficient in specific acoustic noise environments.

4. Internet Can Be Extended Underwater

By transmitting simple text messages, it can be seen that these are, in fact, small packets. IP packets are built from many small packets cemented together. Granted, there are many levels of abstraction that are not being addressed, but with the proper protocols, especially at the MAC and Data Link layers, extension of the Internet to underwater entities is only a matter of time.

5. Where This Is Headed

Standardization is the key to acceptance of any new idea. Just as the wireless and cellular telephone communities are striving for standards so communications can progress into the future, so must the underwater acoustic data communications community. The problem is that there are not really enough users to pressure the groups that have proprietary

systems that are tailored to their specific applications and are working as well as they need them to work. Actually, this process is not only necessary, but inevitable!

B. RECOMMENDATIONS FOR FUTURE WORK

1. Moving Layer Functionality Down Towards Hardware

Let the appropriate layer do the appropriate job. Similar but not necessarily identical to other implementations such as Ethernet. In particular, parameters will be far different.

2. Different Coding

Other coding methods are available, and source code is available for implementation and experimentation. Sources for two Reed-Solomon encoder/decoders are given in the reference section. Another avenue is creation of new coding methods to maximize some particular aspect of FEC. For example, in addition or instead of a block error correction for bit errors, you can recover whole blocks in a tunable way by bitwise exclusive or N blocks to produce a summary block. If you detect losing any one block, you can X-OR all the others with the summary/ checksum' block to recover the missing block. Another design, which was implemented in a prototype but not generalized yet, was to use GNU ecc to encode small blocks for bit errors then wrap blocks in framing (start characters, length, checksum) and X-OR blocks to a tunable number of windowed summary blocks. I.e.: setup X summary blocks. While sending blocks, X-OR each block into the X [current_block_number (N)% num_summary_registers] summary register. You then output the summary blocks every so often and start over. You can tune how often to send the summary blocks and how many there are. This gives you failure recovery for up to X blocks in a row (or in X different slots) every PERIOD blocks. The Reed-Solomon or similar block encoding can help by recovering from N bit flips in each of the blocks, improving the chances of each block surviving. With those parameters you should be able to handle a pretty high level of noise in a tunable way.

The Hamming code used in this thesis created 3 bytes out of 2. However, [Vukadin 91] describes use of a 7,4 hamming code to take each byte, break it up into 2 nibbles, and encode each nibble with the code. Now, each byte will be encoded using 2 7-bit hamming words, making each byte stand alone as a series of one transmission parameter for the Divetracker routine of 2 bytes, then 2 more bytes, etc. Preambles for synchronization are still needed.

3. Necessity of IETF Standards

It cannot be stressed strongly enough that without standardization, compatibility among acoustic devices will never be realized. An open, public dialog in the acoustic telemetry community that includes academics, industry, users and students needs to be started that can lead to a standardization process for IP/SW using the guidelines provided by the Internet Engineering Task Force (IETF) [Stewart 94, IAB 95].

4. FEC Filter to Include Selectable Encoding Algorithms

As the acoustic channel changes, be it in orientation or environment, encoding and decoding methods required for accurate communications will change as well. Some form of feedback to the transmitter will be required to allow the transmitter to modify its encoding algorithm and to let the receiver know that the changes were made so that the decoding algorithm can be changed in turn. This may be hard-coded into the system, or dynamic so changes are able to be made as conditions change.

5. Incorporation Into NPS AUV Architecture

Work has already begun on implementing the DiveTracker system into the NPS *Phoenix* AUV for navigation purposes so that both the AUV operating system and the operating personnel know vehicle location with respect to a baseline. The SMARTDIVE DiveTracker operating system (DTOS) is already programmed to handle a specialized message communications protocol used to send control messages to and from the AUV, but higher level protocols have yet to be implemented both into DTOS and into the strategic

levels of the AUV operating system. Incorporation of a FEC code that meets the requirements set forth in Chapter V pertaining to compatibility with existing protocols is easily done with a protocol that is still very plastic. Enabling the AUV to receive higher-level messages, either from the DiveTracker surface station, another ROV-mounted DiveTracker, or even a diver in the water would be one step closer to achieving IP over Seawater.

6. AOSN and Oceanographic Research

Without reliability, an Autonomous Ocean Sampling Network will fail. Furthermore, any AUV transiting through the area will be at risk of loss. Reliability is needed in such a network and can be obtained through incorporation of FEC into the network communications.

7. Point Sur SOSUS Array

The Naval Postgraduate School has established an Ocean Acoustic Observatory in the Monterey Bay National Marine Sanctuary for the purpose of undersea research. Located immediately South of the Point Sur Light-station, Big Sur, California, the Point Sur Naval Facility was part of the Navy's underwater monitoring network (1958-1986) to acoustically track ships and submarines using horizontal arrays of sound powered hydrophones. The school is preserving the array Terminal Equipment compound, and one horizontal hydrophone array. Though some of the information on the facility remains classified, the undersea hydrophone array can be used to monitor both ship and mammal traffic passing through the Monterey Bay National Marine Sanctuary, and provide unclassified data (single hydrophone timeseries) to both public and private research and educational facilities. In addition, this asset could, if properly utilized, provide an excellent testbed for communications with a submerged platform or even a surface vessel via hydrophones. It is proposed that this type of array could be the access point for pelagic AUVs which are not restricted to near-shore environments to communicate mission status and tasking. For more

information, see <http://www.usw.nps.navy.mil/ptsur.html> or http://www.mbari.org/Ocean_Engineering/95e333.html.

8. Quantitatively Determine Signal to Noise Ratio (SNR)

Noise data from a variety of underwater environments needs to be collected and recorded, then played back at representative signal levels in a test tank or pool while various FEC codes are tested for reliability. Use of Reed-Solomon codes (such as the code in Appendix E), and alternate codes might be tested and incorporated into the NPS AUV system.

9. Implementation of DiveTracker in the WHOI ALAN Project

The Acoustic LAN described in [Catipovic 93] [McMullen 94] was unable to meet expectations because of battery extinction due to continues retransmissions of messages which were never received by the intended receiver. This caused the entire project to be recovered and sent back to Woods Hole Oceanographic Institution. Those problems were essentially the problem statement for this thesis. An excellent follow-on thesis might be the installation of DiveTrackers in the same orientation and tasking as the original ALAN, and by incorporating FEC, determine whether it can be accomplished.

10. Protocol Analysis

[Talavage 94] presents a store-and-forward protocol that could easily be analyzed and implemented in the NPS DiveTracker system. This would make an excellent thesis topic for a network student.

11. Implementation of PPP

Because the Point-to-Point protocol's only requirement is a full-duplex channel (in order to use HDLC), it makes no demands on the modem or RS-232 interface other than Receive and Transmit. This makes it ideal for implementation using a set of 4 acoustic modems to provide duplex channels, or if 2 duplex-capable modems are available [Fischer 93] they can be used as a starting point.

12. Additional Improvements to *AMODEM.DC*

AMODEM.DC's software configuration program needs additional integration between speeds 3 and 4. A speed of 3 seems to be too slow, and a speed of 4 is too fast. At best, this might be a sliding control that is user-configurable from speeds of very slow to a maximum of around 4800 Baud. In addition, as was mentioned before, *AMODEM.DC* would benefit greatly from flow control. Implementation of flow control into *AMODEM.DC* is also work which would provide a networking or software engineering student well.

APPENDIX A: CODE AND SOURCES

Compilation of this code was done at the command line using Borland C++ or Turbo C. The file *PROM_TC.LIB* was put into the \LIB directory, and the file *PROMODEM.H* was put into the \INCLUDE directory of the C compiler. Every compilation was done by invoking the Turbo C compiler as C:\>TCC <FILENAME> PROM_TC.LIB or the Borland C++ compiler as C:\>BCC <FILENAME> PROM_TC.LIB in order to provide the compiler the location of the ProModem library file.

1. HAMMING ENCODER SIMULATION (TRANSMITTER SIDE)

```
/* HAMMSIMM.C Hamming Encoder/DecoderSimulation by Stephen P. Reimers */
/* Code liberally borrowed from TERMINAL.C by Adrian J. Michaud */
/* compiled using Borland C++ at command line */

#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <dos.h>
#include "promodem.h"
float P_single_bit_error;
int Bit_Error_Freq;
int total_errors_injected = 0;
#define DECODED_STR_LEN 52
#define ENCODED_STR_LEN 78
FILE *fp;

void hamming_encoder(char input_string[], char encoded_string[]);
void inject_error(char encoded_string[], char encoded_error_string[]);
void main(void);

void main(void)
{
    char filename[20];
    char main_input_string[DECODED_STR_LEN] = "The purpose of computing is insight,
        not numbers!!";
    char xmit_encoded_string[ENCODED_STR_LEN];
    char xmit_encoded_error_string[ENCODED_STR_LEN];
    int i, count;
    int repetitions;
    char ch;
    AJMS cb1;
    unsigned int irq=0, base_address=0;
    char com_port[80];
    char baud_rate[80];
    long baudRate=0;

    printf("Enter name of record file: ");
    gets(filename);
```

```

if ( ( fp = fopen (filename, "w")) == NULL)
{
    fprintf(stderr, "Error opening file %s. ", filename);
    exit(1);
}
printf("Enter comm port to use: ");
gets(com_port);

printf("Enter baud rate: ");
gets(baud_rate);

printf("Enter number of repetitions: ");
fprintf(fp, "\nEnter number of repetitions: ");
scanf("%d", &repetitions);
fprintf(fp, "%d", repetitions);

printf("Bit error frequency (8 or 16): ");
fprintf(fp, "\nBit error frequency: ");
scanf("%d", &Bit_Error_Freq);
fprintf(fp, "%d", Bit_Error_Freq);

printf("Enter the probability of a single bit error: ");
fprintf(fp, "\nSingle bit error probability: ");
scanf("%f", &P_single_bit_error);
fprintf(fp, "%f\n", P_single_bit_error);

/* initialize arrays with all zeros */
for ( i=0; i < ENCODED_STR_LEN; i++ )
    xmit_encoded_string[i] = 0;

while (!irq && !base_address)
{
    /* printf("Enter Com port [1 or 2]: ");
    gets(com_port); */

    if (strstr(com_port, "1"))
    {
        base_address = 0x3f8;
        irq = 4;
    }

    else
    if (strstr(com_port, "2"))
    {
        base_address = 0x2f8;
        irq = 3;
    }
}

#define BUFFER_SIZE 4096 /* Use a 4K IRQ Buffer */

SetupControlBlock(&cb1 , base_address , irq , BUFFER_SIZE);

if (OpenCom(&cb1) != SUCCESSFUL) /* Open Com Port */
{
    printf("\nCan't Open Com Port!");
    return;
}

while (!baudRate) /* Obtain desired Baud Rate */
{
    /* printf("\nEnter Baud Rate (300,2400,9600,19200,38400,...): ");
    fprintf(fp, "\nEnter Baud Rate: ");
    gets(baud_rate);
    fprintf(fp, "%s", baud_rate); */

    baudRate = atol(baud_rate);
}

```

```

SetBaudRate(&cb1, baudRate); /* Sets up Baud Rate */
SetDataFormat(&cb1, BITS_8|NO_PARITY|STOP_BITS_1); /* 8-N-1 */

if (SetFIFOMode(&cb1) == FIFO_ENABLED) /* Enable 16550 if any */
{
    SetFIFOTriggerLevel(&cb1, FIFO_16_TRIGGER); /* Turn on 16 byte RCVR FIFO */
    printf("\nUART is a 16550! FIFO's Enabled.\n");
}
/* else printf("\nUART is a 16540 or 8250.\n"); */

/* if (GetCTSRTSStatus(&cb1) == ENABLED) */ /* Check if CTS/RTS is on */
/* printf("\nCTS/RTS Hardware Handshaking Enabled!\n"); */
else
    printf("\nCTS/RTS Hardware Handshaking Disabled!\n"); */

/* printf("\nLoad ANSI.SYS if you wish to display ANSI.\n"); */
/* printf("\nPress [ESC] to Exit this Terminal Program..\n"); */ */

/* printf("Input a string up to %d characters long. \n", DECODED_STR_LEN); */
/* gets(main_input_string); */ */

printf("This is the input string: %s \n", main_input_string );
fprintf(fp, "\nThe input string: %s \n", main_input_string );

hamming_encoder(main_input_string, xmit_encoded_string );
/* printf("This is the xmit_encoded_string (w/o errors): %s \n", */
/* xmit_encoded_string ); */ */

for (count = 0; count < repetitions; count++)
{
    printf("Cycle number: %d \n", count+1);
    fprintf(fp, "Cycle number: %d", count+1);
    for ( i=0; i < ENCODED_STR_LEN; i++ )
        xmit_encoded_error_string[i] = 0;

    inject_error ( xmit_encoded_string, xmit_encoded_error_string);
    /* printf("This is the xmit_encoded_string (WITH errors): %s \n", */
    /* xmit_encoded_error_string); */ */

    /* printf("Press any key to transmit the message. \n"); */ */
    /* getch(); */ */
    sleep(3);

    for (i = 0; i < ENCODED_STR_LEN; i++)
    {
        SendCharacter(&cb1, (unsigned char)xmit_encoded_error_string[i]);
        if (kbhit())
        {
            ch = (char) getch();
            if (ch == 27) break;
        }
    }
} /* End of for loop to go repetition times */

CloseCom(&cb1);
DropDTR(&cb1); /* Disable Data Terminal Ready */
printf("Total number of errors injected: %d \n", total_errors_injected);
fprintf(fp, "\nTotal errors injected: %d \n", total_errors_injected);
fclose(fp);
}

void hamming_encoder(char input_string[], char encoded_string[])
{
    unsigned msg_char1, msg_char2;

```

```

unsigned m1a, m2a, m3a, m4a, m5a, m6a, m7a, m8a, p1a, p2a, p3a, p4a;
unsigned m1b, m2b, m3b, m4b, m5b, m6b, m7b, m8b, p1b, p2b, p3b, p4b;

/* long int coded_char1, coded_char2; */
/* char coded_c, coded_d, coded_e; */
unsigned coded_c, coded_d, coded_e;
char cde_string[4];
int i, index, n;

/* initialize cde_string to all zeros. */
for ( i=0; i < 4; i++ )
    cde_string[i] = 0;

n = strlen(input_string);
if ((n%2) != 0)
{
    strcat(input_string, " ");
    n++;
}

/* printf ("The length of the string is %d. \n", n); */

/* parse thru current_string, two characters at a time */
for (i=0; i < n; i=i+2) /* positions 0 to n-1 */
{
    /* msg_char1 is one character, msg_char2 is second character */
    /* now loop through and take apart msg_char1, then msg_char2 */
    /* These are the same 8 bits, but different types. */
    msg_char1 = msg_char2 = 0;
    msg_char1 = (int) input_string[i];
    msg_char2 = (int) input_string[i+1];

    /* printf("Now encoding character1 %d \n", msg_char1); */

    m1a = m2a = m3a = m4a = m5a = m6a = m7a = m8a = 0;
    m1a = (msg_char1 & 1) >> 0;
    m2a = (msg_char1 & 2) >> 1;
    m3a = (msg_char1 & 4) >> 2;
    m4a = (msg_char1 & 8) >> 3;
    m5a = (msg_char1 & 16) >> 4;
    m6a = (msg_char1 & 32) >> 5;
    m7a = (msg_char1 & 64) >> 6;
    m8a = (msg_char1 & 128) >> 7;

    p1a = p2a = p3a = p4a = 0;
    p1a = m1a ^ m2a ^ m4a ^ m5a ^ m7a;
    p2a = m1a ^ m3a ^ m4a ^ m6a ^ m7a;
    p3a = m2a ^ m3a ^ m4a ^ m8a;
    p4a = m5a ^ m6a ^ m7a ^ m8a;

    /* this is where the Hamming code gets inserted */

    m1b = m2b = m3b = m4b = m5b = m6b = m7b = m8b = 0;
    m1b = (msg_char2 & 1) >> 0;
    m2b = (msg_char2 & 2) >> 1;
    m3b = (msg_char2 & 4) >> 2;
    m4b = (msg_char2 & 8) >> 3;
    m5b = (msg_char2 & 16) >> 4;
    m6b = (msg_char2 & 32) >> 5;
    m7b = (msg_char2 & 64) >> 6;
    m8b = (msg_char2 & 128) >> 7;

    p1b = p2b = p3b = p4b = 0;
    p1b = m1b ^ m2b ^ m4b ^ m5b ^ m7b;
    p2b = m1b ^ m3b ^ m4b ^ m6b ^ m7b;
    p3b = m2b ^ m3b ^ m4b ^ m8b;

```

```

p4b = m5b ^ m6b ^ m7b ^ m8b;

/* this is where the Hamming code gets inserted */
/* now create coded_c, d, and e of type char */
coded_c = coded_d = coded_e = 0;
coded_c = ((m8a << 7) + (m7a << 6) + (m6a << 5) + (m5a << 4) +
           (p4a << 3) + (m4a << 2) + (m3a << 1) + m2a);
/* printf("coded_c: %d%d%d%d%d%d%d \n", m8a, m7a, m6a, m5a, p4a, m4a,
m3a, m2a); */
coded_d = ((p3a << 7) + (m1a << 6) + (p2a << 5) + (p1a << 4) +
           (m8b << 3) + (m7b << 2) + (m6b << 1) + m5b);
/* printf("coded_d: %d%d%d%d%d%d%d \n", p3a, m1a, p2a, p1a, m8b, m7b,
m6b, m5b); */
coded_e = ((p4b << 7) + (m4b << 6) + (m3b << 5) + (m2b << 4) +
           (p3b << 3) + (m1b << 2) + (p2b << 1) + p1b);
/* printf("coded_e: %d%d%d%d%d%d%d \n", p4b, m4b, m3b, m2b, p3b, m1b,
p2b, p1b); */

/* create small string cde_string out of the coded_c, d, & e chars */
for ( index = 0; index < 3; index++) /* zero array */
    cde_string[index] = 0;
cde_string[0] = coded_c;
cde_string[1] = coded_d;
cde_string[2] = coded_e;

/* Lets look at our 2 original characters which is now a 3 character string
*/
/* printf("This is our 3 character encoded string: %x %x %x \n",
cde_string[0], cde_string[1], cde_string[2]); */

/* now we build the encoded_string out of the cde_string */
/*first test for an empty string */
if (strlen(encoded_string)==0)
{
    /* printf("The length of string encoded_string is Zero. \n"); */
    strcpy(encoded_string, cde_string);
    /* printf("The encoded_string now looks like this: %s \n",
encoded_string); */
}
else
{
    /* printf("going to the else clause, strlen >0. \n"); */
    strcat (encoded_string, cde_string);
}

} /* end of huge for-loop */

/* printf("the length of encoded_string is %d \n", strlen(encoded_string)); */

/* the string is now encoded. */
}
/* End of Hamming Encoder */

/*****/
void inject_error(char encoded_string[], char encoded_error_string[])
{
    int i, error_bit, error_count=0;
    time_t t;
    /* initialize array */
    for (i = 0; i < ENCODED_STR_LEN ; i++)
        encoded_error_string[i] = 0;
    strcpy( encoded_error_string, encoded_string);

    srand((unsigned) time(&t));
    if (Bit_Error_Freq == 8)
        for (i = 0; i < strlen(encoded_error_string); i++)

```



```

{
    /* printf("i is now: %d \n", i); */
    if (( rand() / (pow(2.0, 15.0) - 1)) < P_single_bit_error)
    {
        /* inject error in a single random bit */

        error_bit = rand() % 8;
        switch (error_bit)
        {
            case 0:
                encoded_error_string[i] = encoded_error_string[i] ^ 1;
                break;
            case 1:
                encoded_error_string[i] = encoded_error_string[i] ^ 2;
                break;
            case 2:
                encoded_error_string[i] = encoded_error_string[i] ^ 4;
                break;
            case 3:
                encoded_error_string[i] = encoded_error_string[i] ^ 8;
                break;
            case 4:
                encoded_error_string[i] = encoded_error_string[i] ^ 16;
                break;
            case 5:
                encoded_error_string[i] = encoded_error_string[i] ^ 32;
                break;
            case 6:
                encoded_error_string[i] = encoded_error_string[i] ^ 64;
                break;
            case 7:
                encoded_error_string[i] = encoded_error_string[i] ^ 128;
                break;
        }
        /* printf("Injected error into character %d bit %d \n", i+1,
error_bit + 1); */
        error_count++;
    }
}

else /* must be every 16 bits, instead of every 8. This eliminates 2 errors
per word. */
{
    for (i = 0; i < strlen(encoded_error_string); i = i+2 )
    {
        if (( rand() / (pow(2.0, 15.0) - 1)) < P_single_bit_error)
        {
            /* inject error in a single random bit */

            error_bit = rand() % 8;
            switch (error_bit)
            {
                case 0:
                    encoded_error_string[i] = encoded_error_string[i] ^ 1;
                    break;
                case 1:
                    encoded_error_string[i] = encoded_error_string[i] ^ 2;
                    break;
                case 2:
                    encoded_error_string[i] = encoded_error_string[i] ^ 4;
                    break;
                case 3:
                    encoded_error_string[i] = encoded_error_string[i] ^ 8;
                    break;
                case 4:
                    encoded_error_string[i] = encoded_error_string[i] ^ 16;
                    break;
                case 5:

```

```

        encoded_error_string[i] = encoded_error_string[i] ^ 32;
        break;
    case 6:
        encoded_error_string[i] = encoded_error_string[i] ^ 64;
        break;
    case 7:
        encoded_error_string[i] = encoded_error_string[i] ^ 128;
        break;
    }
    /* printf("Injected error into character %d bit %d \n", i+1,
error_bit + 1); */
    error_count++;
}
}

printf("Errors injected: %d \n", error_count);
fprintf(fp, "\nErrors injected: %d \n", error_count);
total_errors_injected = total_errors_injected + error_count;
}

/*****

```

2. HAMMING DECODER SIMULATION (RECEIVER SIDE)

```

/* RCVRSIMM.C Receiver Simulator by Stephen P. Reimers */
/* Serial Code liberally borrowed from */
/* TERMINAL.C by Adrian J. Michaud */

#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "promodem.h"
#define DECODED_STR_LEN 52
#define ENCODED_STR_LEN 78
int total_errors_corrected = 0;

void hamming_decoder(char encoded_string[], char error_string[], char
    decoded_string[]);
void clear_kb(void);
void main(void);
FILE *fp;

void main(void)
{
    char filename[20];
    char rcvd_encoded_string[ENCODED_STR_LEN];
    char rcvd_decoded_string[DECODED_STR_LEN];
    char corrected_output_string[DECODED_STR_LEN];

    int count = 0;
    int i, a;
    int repetitions;
    char ch;
    AJMS cbi;
    unsigned int irq=0, base_address=0;
    char com_port[80];
    char baud_rate[80];
    long baudRate=0;

    printf("Enter name of record file: ");
    gets(filename);
    if ( (fp = fopen(filename, "w")) == NULL)
    {
        fprintf(stderr, "Error opening file %s.", filename);
        exit(1);
    }
}

```

```

    }
    printf("Enter comm port to use: ");
    gets(com_port);
    printf("Enter baud rate: ");
    fprintf(fp, "\nEnter baud rate: ");
    gets(baud_rate);
    fprintf(fp, "%s", baud_rate);
    printf("Enter number of repetitions: ");
    fprintf(fp, "\nEnter number of repetitions: ");
    scanf("%d", &repetitions);
    fprintf(fp, "%d", repetitions);

    for (count = 0; count < repetitions; count++)
    {
        printf("Cycle number: %d \n", count+1);
        fprintf(fp, "\nCycle number: %d \n", count+1);

        /* initialize variables */
        for ( i=0; i < ENCODED_STR_LEN; i++ )
            rcvd_encoded_string[i] = 0;

        for ( i=0; i < DECODED_STR_LEN; i++ )
        {
            rcvd_decoded_string[i] = 0;
            corrected_output_string[i] = 0;
        }

        while (!irq && !base_address)
        {
            if (strstr(com_port, "1"))
            {
                base_address = 0x3f8;
                irq = 4;
            }

            else
            if (strstr(com_port, "2"))
            {
                base_address = 0x2f8;
                irq = 3;
            }
        }

#define BUFFER_SIZE 4096 /* Use a 4K IRQ Buffer */

        SetupControlBlock(&cb1 , base_address , irq , BUFFER_SIZE);

        if (OpenCom(&cb1) != SUCCESSFUL) /* Open Com Port */
        {
            printf("\nCan't Open Com Port!");
            return;
        }

        while (!baudRate) /* Obtain desired Baud Rate */
        {
            baudRate = atol(baud_rate);
        }

        SetBaudRate(&cb1, baudRate); /* Sets up Baud Rate */
        SetDataFormat(&cb1, BITS_8|NO_PARITY|STOP_BITS_1); /* 8-N-1
*/

        if (SetFIFOMode(&cb1) == FIFO_ENABLED) /* Enable 16550 if any */
        {
            SetFIFOTriggerLevel(&cb1, FIFO_16_TRIGGER); /* Turn on 16 byte RCVR FIFO
*/
            printf("\nUART is a 16550! FIFO's Enabled.\n");
        }
    }

```

```

/* else printf("\nUART is a 16540 or 8250.\n"); */

if (GetCTSRTSStatus(&cb1) == ENABLED) /* Check if CTS/RTS is on */
    printf("CTS/RTS Hardware Handshaking Enabled!\n");
else
    printf("\nCTS/RTS Hardware Handshaking Disabled!\n");

/* printf("\nLoad ANSI.SYS if you wish to display ANSI.\n");
   printf("\nPress [ESC] to Exit this Terminal Program..\n"); */

for ( i=0; i < ENCODED_STR_LEN; i++ ) /* Loop to fill string */
{
    if (kbhit())
    {
        ch = (char)getch();
        if (ch == 27)
            return;
    }

    while (CheckQueue(&cb1) != CHARACTERS_WAITING) /* Check IRQ buffer for
chars */
        ;

    rcvd_encoded_string[i] = GetCharacter(&cb1);
    printf(".");

    /* printf("exited while loop! \n"); */
}

    hamming_decoder(rcvd_encoded_string, rcvd_decoded_string,
corrected_output_string);

    printf("The received decoded string: %s \n", rcvd_decoded_string );
    fprintf(fp, "The received decoded string: %s", rcvd_decoded_string );

    printf("The corrected decoded string: %s \n", corrected_output_string );
    fprintf(fp, "\nThe corrected decoded string: %s \n", corrected_output_string
);

    CloseCom(&cb1);
    DropDTR(&cb1); /* Disable Data Terminal Ready */

    /* printf("we've reached CloseCom and DropDTR. \n"); */
} /* end big for-loop */
printf("Total errors corrected: %d \n", total_errors_corrected);
fprintf(fp, "\nTotal errors corrected: %d \n", total_errors_corrected);
fclose(fp);
printf("just before final brace. \n");

) /* End of Main */

/*Hamming 12/8 Decoder */

void hamming_decoder(char encoded_string[], char error_string[], char
decoded_string[])
{
    unsigned msg_char1, msg_char2;
    unsigned m1a, m2a, m3a, m4a, m5a, m6a, m7a, m8a, p1a, p2a, p3a, p4a;
    unsigned m1b, m2b, m3b, m4b, m5b, m6b, m7b, m8b, p1b, p2b, p3b, p4b;
    unsigned syndrome1, syndrome2, new_syndrome1, new_syndrome2;
    unsigned new_p1a = 0, new_p2a = 0, new_p3a = 0, new_p4a = 0;
    unsigned new_p1b = 0, new_p2b = 0, new_p3b = 0, new_p4b = 0;
    unsigned coded_c, coded_d, coded_e;
    char a_string[2], b_string[2];
    unsigned error_location1, error_location2;
    int i, b, n, error_count = 0;
    int index = 0;

```

```

/* now get the encoded_string, and start decoding it */
/* printf("the encoded_string: %s \n", encoded_string); */

n = strlen(encoded_string);
/* printf("length of encoded_string is %d \n", n); */

if ((n%3) != 0)
{
    fprintf(fp, "\nWARNING: encoded string is not evenly divisible by 3! \n");
    printf("WARNING: encoded string is not evenly divisible by 3! \n");
    if ((n%3) == 1)
    {
        strcat( encoded_string, "  "); /*append 2 spaces*/
        n = n+2;
    }
    else /*n%3 must equal 2, so we only need one space */
    {
        strcat( encoded_string, " ");
        n++;
    }
} /*end of if clause */

/* parse thru output_string, three characters at a time */
for (i = 0; i < n; i = i+3) /* positions 0 to n-1 */
{
    /* initialize both strings to all zeros. */
    for ( b = 0; b < 2; b++ )
        a_string[b] = 0;
    for ( b = 0; b < 2; b++ )
        b_string[b] = 0;
    /* printf("this is the nulled-out a_string: %s \n", a_string);
    printf("this is the nulled-out b_string: %s \n", b_string); */

    coded_c = (int) encoded_string[i];
    coded_d = (int) encoded_string[i+1];
    coded_e = (int) encoded_string[i+2];

    /* now we have to get 12 bits back out of the 3 chars, which are the
    original message bits plus the parity bits. */
    m1a = m2a = m3a = m4a = m5a = m6a = m7a = m8a = 0;
    p1a = p2a = p3a = p4a = p1b = p2b = p3b = p4b = 0;
    m1b = m2b = m3b = m4b = m5b = m6b = m7b = m8b = 0;

    m8a = (coded_c & 128) >> 7;
    m7a = (coded_c & 64) >> 6;
    m6a = (coded_c & 32) >> 5;
    m5a = (coded_c & 16) >> 4;
    p4a = (coded_c & 8) >> 3;
    m4a = (coded_c & 4) >> 2;
    m3a = (coded_c & 2) >> 1;
    m2a = (coded_c & 1) >> 0;
    p3a = (coded_d & 128) >> 7;
    m1a = (coded_d & 64) >> 6;
    p2a = (coded_d & 32) >> 5;
    p1a = (coded_d & 16) >> 4;

    m8b = (coded_d & 8) >> 3;
    m7b = (coded_d & 4) >> 2;
    m6b = (coded_d & 2) >> 1;
    m5b = (coded_d & 1) >> 0;
    p4b = (coded_e & 128) >> 7;
    m4b = (coded_e & 64) >> 6;
    m3b = (coded_e & 32) >> 5;
    m2b = (coded_e & 16) >> 4;
    p3b = (coded_e & 8) >> 3;
    m1b = (coded_e & 4) >> 2;

```

```

p2b = (coded_e & 2) >> 1;
p1b = (coded_e & 1) >> 0;

msg_char1 = msg_char2 = 0;
msg_char1 = ((m8a << 7) + (m7a << 6) + (m6a << 5) + (m5a << 4) +
(m4a << 3) + (m3a << 2) + (m2a << 1) + m1a);

msg_char2 = ((m8b << 7) + (m7b << 6) + (m6b << 5) + (m5b << 4) +
(m4b << 3) + (m3b << 2) + (m2b << 1) + m1b);

/* printf("The two characters decoded are: %c, %c. \n", msg_char1,
msg_char2); */

/* Create the string of UNCORRECTED incoming characters. */

error_string[index] = msg_char1;
error_string[index+1] = msg_char2;
index = (index +2);

/*create the syndrome word for the characters received */
syndrome1 = syndrome2 = 0;
syndrome1 = ((p4a << 3) + (p3a << 2) + (p2a << 1) + p1a);
/* printf("This is received syndrome1: %x \n", syndrome1); */

syndrome2 = ((p4b << 3) + (p3b << 2) + (p2b << 1) + p1b);
/* printf("This is received syndrome2: %x \n", syndrome2); */

/* recompute the parity bits from the received characters */

m1a = m2a = m3a = m4a = m5a = m6a = m7a = m8a = 0;
new_p1a = new_p2a = new_p3a = new_p4a = 0;
new_p1b = new_p2b = new_p3b = new_p4b = 0;
m1b = m2b = m3b = m4b = m5b = m6b = m7b = m8b = 0;

m1a = (msg_char1 & 1) >> 0;
m2a = (msg_char1 & 2) >> 1;
m3a = (msg_char1 & 4) >> 2;
m4a = (msg_char1 & 8) >> 3;
m5a = (msg_char1 & 16) >> 4;
m6a = (msg_char1 & 32) >> 5;
m7a = (msg_char1 & 64) >> 6;
m8a = (msg_char1 & 128) >> 7;

new_p1a = m1a ^ m2a ^ m4a ^ m5a ^ m7a;
new_p2a = m1a ^ m3a ^ m4a ^ m6a ^ m7a;
new_p3a = m2a ^ m3a ^ m4a ^ m8a;
new_p4a = m5a ^ m6a ^ m7a ^ m8a;

m1b = (msg_char2 & 1) >> 0;
m2b = (msg_char2 & 2) >> 1;
m3b = (msg_char2 & 4) >> 2;
m4b = (msg_char2 & 8) >> 3;
m5b = (msg_char2 & 16) >> 4;
m6b = (msg_char2 & 32) >> 5;
m7b = (msg_char2 & 64) >> 6;
m8b = (msg_char2 & 128) >> 7;

new_p1b = m1b ^ m2b ^ m4b ^ m5b ^ m7b;
new_p2b = m1b ^ m3b ^ m4b ^ m6b ^ m7b;
new_p3b = m2b ^ m3b ^ m4b ^ m8b;
new_p4b = m5b ^ m6b ^ m7b ^ m8b;

new_syndrome1 = new_syndrome2 = 0;
new_syndrome1 = ((new_p4a << 3) + (new_p3a << 2) + (new_p2a << 1) +
new_p1a);
/* printf("This is new_syndrome1: %x \n", new_syndrome1); */

```

```

new_syndrome2 = ((new_p4b << 3) + (new_p3b << 2) + (new_p2b << 1) +
new_p1b);
/* printf("This is new_syndrome2: %x \n", new_syndrome2); */

/*now check to see if the syndrome words are the same. If not, perform
bit-flipping on bit in error. */

if ((new_syndrome1 ^ syndrome1) != 0)
{
    error_location1 = (((new_p4a ^ p4a) << 3) + ((new_p3a ^ p3a) << 2) +
((new_p2a ^ p2a) << 1) + (new_p1a ^ p1a));
    /* printf("Error at location %d. \n", error_location1); */
    error_count++;
    switch (error_location1)
    {
        case 1:
        {
            p1a = (p1a ^ 1);
            break;
        }
        case 2:
        {
            p2a = (p2a ^ 1);
            break;
        }
        case 3:
        {
            m1a = (m1a ^ 1);
            break;
        }
        case 4:
        {
            p3a = (p3a ^ 1);
            break;
        }
        case 5:
        {
            m2a = (m2a ^ 1);
            break;
        }
        case 6:
        {
            m3a = (m3a ^ 1);
            break;
        }
        case 7:
        {
            m4a = (m4a ^ 1);
            break;
        }
        case 8:
        {
            p4a = (p4a ^ 1);
            break;
        }
        case 9:
        {
            m5a = (m5a ^ 1);
            break;
        }
        case 10:
        {
            m6a = (m6a ^ 1);
            break;
        }
        case 11:
        {

```

```

        m7a = (m7a ^ 1);
        break;
    }
    case 12:
    {
        m8a = (m8a ^ 1);
        break;
    }
    default:
    {
        printf("something is going wrong in case for syndrome1. \n");
    }
} /* end of switch statement. */

msg_char1 = 0;
msg_char1 = ((m8a << 7) + (m7a << 6) + (m6a << 5) + (m5a << 4) +
              (m4a << 3) + (m3a << 2) + (m2a << 1) + m1a);
/* printf("corrected character is: %c \n", msg_char1); */
}
/* printf("we made it to the end of the if block. \n"); */

b_string[0] = msg_char1;
if (strlen(decoded_string) == 0)
{
    strcpy(decoded_string, b_string);
}
else
{
    strcat(decoded_string, b_string);
}

if ((new_syndrome2 ^ syndrome2) != 0)
{
    error_location2 = (((new_p4b ^ p4b) << 3) + ((new_p3b ^ p3b) << 2) +
                      ((new_p2b ^ p2b) << 1) + (new_p1b ^ p1b));
    /* printf("Error at location %d. \n", error_location2); */
    error_count++;

    switch (error_location2)
    {
        case 1:
        {
            p1b = (p1b ^ 1);
            break;
        }
        case 2:
        {
            p2b = (p2b ^ 1);
            break;
        }
        case 3:
        {
            m1b = (m1b ^ 1);
            break;
        }
        case 4:
        {
            p3b = (p3b ^ 1);
            break;
        }
        case 5:
        {
            m2b = (m2b ^ 1);
            break;
        }
        case 6:

```



```

        {
            m3b = (m3b ^ 1);
            break;
        }
    case 7:
        {
            m4b = (m4b ^ 1);
            break;
        }
    case 8:
        {
            p4b = (p4b ^ 1);
            break;
        }
    case 9:
        {
            m5b = (m5b ^ 1);
            break;
        }
    case 10:
        {
            m6b = (m6b ^ 1);
            break;
        }
    case 11:
        {
            m7b = (m7b ^ 1);
            break;
        }
    case 12:
        {
            m8b = (m8b ^ 1);
            break;
        }
    default:
        {
            printf("something is going wrong in case for syndrome2. \n");
        }
    } /* end switch block */

    msg_char2 = ((m8b << 7) + (m7b << 6) + (m6b << 5) + (m5b << 4) +
                (m4b << 3) + (m3b << 2) + (m2b << 1) + m1b);
    /* printf("corrected character is: %c \n", msg_char2); */
}

a_string[0] = msg_char2;
if (strlen(decoded_string) == 0)
{
    strcpy(decoded_string, a_string);
}
else
{
    strcat(decoded_string, a_string);
}

}
printf("Error count for string: %d \n", error_count);
fprintf(fp, "Error count: %d\n", error_count);
total_errors_corrected = total_errors_corrected + error_count;
} /* End function. */

```

3. UNCODED ASCII STRING DIVETRACKER TRANSMITTER

/* TRAN.C Transmitter Simulator by Stephen P. Reimers */

```

/* Serial Code liberally borrowed from */
/* TERMINAL.C by Adrian J. Michaud */

/* Compiled using Borland C++ at command line */

#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include "promodem.h"
#define DECODED_STR_LEN 52

FILE *fp;

void main()
{
    char filename[20];
    char main_input_string[DECODED_STR_LEN] = "The purpose of computing is insight,
    not numbers!!";
    int i, count;
    int repetitions;
    char ch;
    AJMS cb1;
    unsigned int irq=0, base_address=0;
    char com_port[80] = "1";
    char baud_rate[80] = "4800";
    long baudRate=0;
    int delay1, delay2;

    printf("Enter name of record file: ");
    gets ( filename );
    if ( ( fp = fopen (filename, "w") ) == NULL)
    {
        fprintf(stderr, "Error opening file %s. ", filename);
        exit(1);
    }
    /* printf("Enter comm port to use: ");
    gets(com_port);

    printf("Enter baud rate: ");
    gets(baud_rate); */

    printf("Enter number of repetitions: ");
    fprintf(fp, "\nEnter number of repetitions: ");
    scanf("%d", &repetitions);
    fprintf(fp, "%d", repetitions);

    while (!irq && !base_address)
    {
        if (strstr(com_port, "1"))
        {
            base_address = 0x3f8;
            irq = 4;
        }
        else
        if (strstr(com_port, "2"))
        {
            base_address = 0x2f8;
            irq = 3;
        }
    }

#define BUFFER_SIZE 4096 /* Use a 4K IRQ Buffer */
    SetupControlBlock(&cb1, base_address, irq, BUFFER_SIZE);
    if (OpenCom(&cb1) != SUCCESSFUL) /* Open Com Port */
    {
        printf("\nCan't Open Com Port!");
    }
}

```

```

    return;
}

while (!baudRate) /* Obtain desired Baud Rate */
{
    baudRate = atol(baud_rate);
}

SetBaudRate(&cb1, baudRate); /* Sets up Baud Rate */
SetDataFormat(&cb1, BITS_8|NO_PARITY|STOP_BITS_1); /* 8-N-1 */

if (SetFIFOMode(&cb1) == FIFO_ENABLED) /* Enable 16550 if any */
{
    SetFIFOTriggerLevel(&cb1, FIFO_16_TRIGGER); /* Turn on 16 byte RCVR FIFO */
    printf("\nUART is a 16550! FIFO's Enabled.\n");
}
else printf("\nUART is a 16540 or 8250.\n");

if (GetCTSRTSStatus(&cb1) == ENABLED) /* Check if CTS/RTS is on */
    printf("\nCTS/RTS Hardware Handshaking Enabled!\n");
else
    printf("\nCTS/RTS Hardware Handshaking Disabled!\n");

printf("\nPress [ESC] to Exit this Terminal Program..\n");

printf("This is the input string: %s \n", main_input_string);
fprintf(fp, "\nThe input string: %s \n", main_input_string);

for (count = 0; count < repetitions; count++)
{
    printf("Cycle number: %d \n", count+1);
    fprintf(fp, "Cycle number: %d\n", count+1);
    sleep(1);

    for (i = 0; i < DECODED_STR_LEN; i=i+2)
    {
        SendCharacter(&cb1, (unsigned char)main_input_string[i]);
        SendCharacter(&cb1, (unsigned char)main_input_string[i+1]);
        for (delay1=0; delay1 < 1500; delay1++)
        {
            for (delay2=0; delay2 < 500; delay2++)
            {
            };
        }
        if(kbhit())
        {
            ch = (char) getch();
            if (ch == 27) exit(1);
        }
    }
} /* End of for loop to go repetition times */

CloseCom(&cb1);
DropDTR(&cb1); /* Disable Data Terminal Ready */
fclose(fp);
}

```

4. UNCODED ASCII STRING DIVETRACKER RECEIVER

```
/* RCVR.C by Stephen P. Reimers */
/* Uses PROMODEM SERIAL LIBRARY by Adrian J. Michaud */

#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "promodem.h"
#define DECODED_STR_LEN 52

void clear_kb(void);
void main(void);
FILE *fp;

void main(void)
{
    char filename[20];
    char rcvd_string[DECODED_STR_LEN];

    int count = 0;
    int i, a;
    int repetitions;
    char ch;
    AJMS cb1;
    unsigned int irq=0, base_address=0;
    char com_port[80] = "1";
    char baud_rate[80] = "4800";
    long baudRate=0;

    printf("Enter name of record file: ");
    gets(filename);
    if ( (fp = fopen(filename, "w")) == NULL)
    {
        fprintf(stderr, "Error opening file %s.", filename);
        exit(1);
    }
    /* printf("Enter comm port to use: ");
    gets(com_port);
    printf("Enter baud rate: ");
    fprintf(fp, "\nEnter baud rate: ");
    gets(baud_rate);
    fprintf(fp, "%s", baud_rate); */
    printf("Enter number of repetitions: ");
    fprintf(fp, "\nEnter number of repetitions: ");
    scanf("%d", &repetitions);
    fprintf(fp, "%d", repetitions);

    for (count = 0; count < repetitions; count++)
    {
        printf("Cycle number: %d \n", count+1);
        fprintf(fp, "\nCycle number: %d \n", count+1);

        /* initialize variables */
        for ( i=0; i < DECODED_STR_LEN; i++ )
            rcvd_string[i] = 0;
        while (!irq && !base_address)
        {
            if (strstr(com_port, "1"))
            {
                base_address = 0x3f8;
                irq = 4;
            }
            else

```

```

        if (strstr(com_port, "2"))
        {
            base_address = 0x2f8;
            irq           = 3;
        }
    }

#define BUFFER_SIZE 4096 /* Use a 4K IRQ Buffer */

    SetupControlBlock(&cb1, base_address, irq, BUFFER_SIZE);

    if (OpenCom(&cb1) != SUCCESSFUL) /* Open Com Port */
    {
        printf("\nCan't Open Com Port!");
        return;
    }

    while (!baudRate) /* Obtain desired Baud Rate */
    {
        baudRate = atol(baud_rate);
    }

    SetBaudRate(&cb1, baudRate); /* Sets up Baud Rate */
    SetDataFormat(&cb1, BITS_8|NO_PARITY|STOP_BITS_1); /* 8-N-1 */

    if (SetFIFOMode(&cb1) == FIFO_ENABLED) /* Enable 16550 if any */
    {
        SetFIFOTriggerLevel(&cb1, FIFO_16_TRIGGER); /* Turn on 16 byte RCVR FIFO
*/
        printf("\nUART is a 16550! FIFO's Enabled.\n");
    }
    /* else printf("\nUART is a 16540 or 8250.\n"); */

    if (GetCTSRTSStatus(&cb1) == ENABLED) /* Check if CTS/RTS is on */
        printf("CTS/RTS Hardware Handshaking Enabled!\n");
    else
        printf("\nCTS/RTS Hardware Handshaking Disabled!\n");

    /* printf("\nLoad ANSI.SYS if you wish to display ANSI.\n");
    printf("\nPress (ESC) to Exit this Terminal Program..\n"); */

    for (i=0; i < DECODED_STR_LEN; i++) /* Loop to fill string */
    {
        if (kbhit())
        {
            ch = (char)getch();
            if (ch == 27)
                return;
        }

        while (CheckQueue(&cb1) != CHARACTERS_WAITING) /* Check IRQ buffer for
chars */
            ;

        rcvd_string[i] = GetCharacter(&cb1);
        printf(".");

        /* printf("exited while loop! \n"); */
    }

    printf("The received string: %s \n", rcvd_string);
    fprintf(fp, "The received string: %s", rcvd_string);
    CloseCom(&cb1);
    DropDTR(&cb1); /* Disable Data Terminal Ready */

    /* printf("we've reached CloseCom and DropDTR. \n"); */
} /* end big for-loop */
fclose(fp);

```

```
printf("just before final brace. \n");
} /* End of Main */
```

5. HAMMING ENCODER DIVETRACKER TRANSMITTER

```
/* Hamming Encoder by Stephen P. Reimers */
/* No error injection routine - errors are from acoustic channel */
/* Code liberally borrowed from TERMINAL.C by Adrian J. Michaud */
/* Compiled using Borland C++ at command line */

#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <dos.h>
#include "promodem.h"
#define DECODED_STR_LEN 52
#define ENCODED_STR_LEN 76

FILE *fp;

void hamming_encoder(char input_string[], char encoded_string[]);
void main(void);

void main(void)
{
    char filename[20];
    char main_input_string[DECODED_STR_LEN] = "The purpose of computing is insight,
    not numbers!!";
    char xmit_encoded_string[ENCODED_STR_LEN];
    int i, count;
    int repetitions;
    char ch;
    AJMS cbl;
    unsigned int irq=0, base_address=0;
    char com_port[80] = "1";
    char baud_rate[80] = "4800";
    long baudRate=0;
    int delay1, delay2;

    printf("Enter name of record file: ");
    gets(filename);
    if ( ( fp = fopen (filename, "w") ) == NULL) .
    {
        fprintf(stderr, "Error opening file %s. ", filename);
        exit(1);
    }

    printf("Enter number of repetitions: ");
    fprintf(fp, "\nEnter number of repetitions: ");
    scanf("%d", &repetitions);
    fprintf(fp, "%d", repetitions);

    /* initialize arrays with all zeros */
    for ( i=0; i < ENCODED_STR_LEN; i++ )
        xmit_encoded_string[i] = 0;

    while (!irq && !base_address)
    {
        /* printf("Enter Com port [1 or 2]: ");
        gets(com_port); */

        if (strstr(com_port, "1"))
        {
```

```

        base_address = 0x3f8;
        irq          = 4;
    }

    else
    {
        if (strstr(com_port, "2"))
        {
            base_address = 0x2f8;
            irq          = 3;
        }
    }

#define BUFFER_SIZE 4096 /* Use a 4K IRQ Buffer */

SetupControlBlock(&cb1, base_address, irq, BUFFER_SIZE);

if (OpenCom(&cb1) != SUCCESSFUL) /* Open Com Port */
{
    printf("\nCan't Open Com Port!");
    return;
}

baudRate = atol(baud_rate);
SetBaudRate(&cb1, baudRate); /* Sets up Baud Rate */
SetDataFormat(&cb1, BITS_8|NO_PARITY|STOP_BITS_1); /* 8-N-1 */

if (SetFIFOMode(&cb1) == FIFO_ENABLED) /* Enable 16550 if any */
{
    SetFIFOTriggerLevel(&cb1, FIFO_16_TRIGGER); /* Turn on 16 byte RCVR FIFO */
    printf("\nUART is a 16550! FIFO's Enabled.\n");
}
/* else printf("\nUART is a 16540 or 8250.\n"); */

/* if (GetCTSRTSStatus(&cb1) == ENABLED) */ /* Check if CTS/RTS is on */
/* printf("\nCTS/RTS Hardware Handshaking Enabled!\n"); */
else
    printf("\nCTS/RTS Hardware Handshaking Disabled!\n"); /*

/* printf("\nLoad ANSI.SYS if you wish to display ANSI.\n"); */
printf("\nPress [ESC] to Exit this Terminal Program..\n"); /*

printf("This is the input string: %s \n", main_input_string);
fprintf(fp, "The input string: %s \n", main_input_string);

hamming_encoder(main_input_string, xmit_encoded_string);
/* printf("This is the xmit_encoded_string (w/o errors): %s \n",
    xmit_encoded_string ); */

for (count = 0; count < repetitions; count++)
{
    printf("Cycle number: %d \n", count+1);
    fprintf(fp, "Cycle number: %d", count+1);

    sleep(3);

    for (i = 0; i < ENCODED_STR_LEN; i=i+2)
    {
        SendCharacter(&cb1, (unsigned char)xmit_encoded_string[i]);
        SendCharacter(&cb1, (unsigned char)xmit_encoded_string[i+1]);
        for (delay1=0; delay1 < 1500; delay1++)
        {
            for (delay2 = 0; delay2 < 500; delay2++)
            {
                ;
            }
        }
        if (kbhit())
        {

```

```

        ch = (char) getch();
        if (ch == 27) exit(1);
    }
} /* End of for loop to go repetition times */

CloseCom(&cb1);
DropDTR(&cb1); /* Disable Data Terminal Ready */
fclose(fp);
}

void hamming_encoder(char input_string[], char encoded_string[])
{
    unsigned msg_char1, msg_char2;
    unsigned m1a, m2a, m3a, m4a, m5a, m6a, m7a, m8a, p1a, p2a, p3a, p4a;
    unsigned m1b, m2b, m3b, m4b, m5b, m6b, m7b, m8b, p1b, p2b, p3b, p4b;

    /* long int coded_char1, coded_char2; */
    /* char coded_c, coded_d, coded_e; */
    unsigned coded_c, coded_d, coded_e;
    char cde_string[4];
    int i, index, n;

    /* initialize cde_string to all zeros. */
    for (i=0; i < 4; i++)
        cde_string[i] = 0;

    n = strlen(input_string);
    if ((n%2) != 0)
    {
        strcat(input_string, " ");
        n++;
    }

    /* printf ("The length of the string is %d. \n", n); */

    /* parse thru current_string, two characters at a time */
    for (i=0; i < n; i=i+2) /* positions 0 to n-1 */
    {
        /* msg_char1 is one character, msg_char2 is second character */
        /* now loop through and take apart msg_char1, then msg_char2 */
        /* These are the same 8 bits, but different types. */
        msg_char1 = msg_char2 = 0;
        msg_char1 = (int) input_string[i];
        msg_char2 = (int) input_string[i+1];

        /* printf("Now encoding character1 %d \n", msg_char1); */

        m1a = m2a = m3a = m4a = m5a = m6a = m7a = m8a = 0;
        m1a = (msg_char1 & 1) >> 0;
        m2a = (msg_char1 & 2) >> 1;
        m3a = (msg_char1 & 4) >> 2;
        m4a = (msg_char1 & 8) >> 3;
        m5a = (msg_char1 & 16) >> 4;
        m6a = (msg_char1 & 32) >> 5;
        m7a = (msg_char1 & 64) >> 6;
        m8a = (msg_char1 & 128) >> 7;

        p1a = p2a = p3a = p4a = 0;
        p1a = m1a ^ m2a ^ m4a ^ m5a ^ m7a;
        p2a = m1a ^ m3a ^ m4a ^ m6a ^ m7a;
        p3a = m2a ^ m3a ^ m4a ^ m8a;
        p4a = m5a ^ m6a ^ m7a ^ m8a;
    }
}

```



```

/* this is where the Hamming code gets inserted */
/* coded_char1 = 0; */
/* coded_char1 = ( p1a + (p2a << 1) + (m1a << 2) + (p3a << 3) +
(m2a << 4) + (m3a << 5) + (m4a << 6) + (p4a << 7) +
(m5a << 8) + (m6a << 9) + (m7a << 10) + (m8a << 11)); */

/* printf("This is the encoded first char: %x \n", coded_char1); */
/* printf("The first character with parity bits added is now:
%d%d%d%d%d%d%d%d \n", m8a, m7a, m6a, m5a, p4a, m4a, m3a, m2a, p3a,
m1a, p2a, p1a); */

/* printf("Now encoding character2 %d \n", msg_char2); */

m1b = m2b = m3b = m4b = m5b = m6b = m7b = m8b = 0;
m1b = (msg_char2 & 1) >> 0;
m2b = (msg_char2 & 2) >> 1;
m3b = (msg_char2 & 4) >> 2;
m4b = (msg_char2 & 8) >> 3;
m5b = (msg_char2 & 16) >> 4;
m6b = (msg_char2 & 32) >> 5;
m7b = (msg_char2 & 64) >> 6;
m8b = (msg_char2 & 128) >> 7;

p1b = p2b = p3b = p4b = 0;
p1b = m1b ^ m2b ^ m4b ^ m5b ^ m7b;
p2b = m1b ^ m3b ^ m4b ^ m6b ^ m7b;
p3b = m2b ^ m3b ^ m4b ^ m8b;
p4b = m5b ^ m6b ^ m7b ^ m8b;

/* this is where the Hamming code gets inserted */
/* coded_char2 = 0; */
/* coded_char2 = ( p1b + (p2b << 1) + (m1b << 2) + (p3b << 3) +
(m2b << 4) + (m3b << 5) + (m4b << 6) + (p4b << 7) +
(m5b << 8) + (m6b << 9) + (m7b << 10) + (m8b << 11)); */

/* printf("This is the encoded second character: %x \n", coded_char2); */
/* printf("The second character with parity bits added is now:
%d%d%d%d%d%d%d%d \n", m8b, m7b, m6b, m5b, p4b, m4b, m3b, m2b, p3b,
m1b, p2b, p1b); */

/* now create coded_c, d, and e of type char */
coded_c = coded_d = coded_e = 0;
coded_c = ((m8a << 7) + (m7a << 6) + (m6a << 5) + (m5a << 4) +
(p4a << 3) + (m4a << 2) + (m3a << 1) + m2a);
/* printf("coded_c: %d%d%d%d%d%d \n", m8a, m7a, m6a, m5a, p4a, m4a,
m3a, m2a); */
coded_d = ((p3a << 7) + (m1a << 6) + (p2a << 5) + (p1a << 4) +
(m8b << 3) + (m7b << 2) + (m6b << 1) + m5b);
/* printf("coded_d: %d%d%d%d%d%d \n", p3a, m1a, p2a, p1a, m8b, m7b,
m6b, m5b); */
coded_e = ((p4b << 7) + (m4b << 6) + (m3b << 5) + (m2b << 4) +
(p3b << 3) + (m1b << 2) + (p2b << 1) + p1b);
/* printf("coded_e: %d%d%d%d%d%d \n", p4b, m4b, m3b, m2b, p3b, m1b,
p2b, p1b); */
/* printf("the coded_c, coded_d and coded_e chars are here: \n %x %x
%x \n ", coded_c, coded_d, coded_e); */

/* create small string cde_string out of the coded_c, d, & e chars */
for ( index = 0; index < 3; index++) /* zero array */
cde_string[index] = 0;
cde_string[0] = coded_c;
cde_string[1] = coded_d;
cde_string[2] = coded_e;

/* Lets look at our 2 original characters which is now a 3 character string

```

```

*/
/* printf("This is our 3 character encoded string: %x %x %x \n",
cde_string[0], cde_string[1], cde_string[2]); */

/* now we build the encoded_string out of the cde_string */
/*first test for an empty string */
if (strlen(encoded_string)==0)
{
    /* printf("The length of string encoded_string is Zero. \n"); */
    strcpy(encoded_string, cde_string);
    /* printf("The encoded_string now looks like this: %s \n",
encoded_string); */
}
else
{
    /* printf("going to the else clause, strlen >0. \n"); */
    strcat (encoded_string, cde_string);
}

} /* end of huge for-loop */

/* printf("the length of encoded_string is %d \n", strlen(encoded_string)); */

/* the string is now encoded. */
}
/* End of Hamming Encoder */

```

6. HAMMING DECODER DIVETRACKER RECEIVER

```

/* HAMMRCVR.C Hamming DECODER by Stephen P. Reimers */
/* Serial code liberally borrowed from */
/* TERMINAL.C by Adrian J. Michaud */

#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "promodem.h"
#define DECODED_STR_LEN 52
#define ENCODED_STR_LEN 76
int total_errors_corrected = 0;

void hamming_decoder(char encoded_string[], char error_string[], char
    decoded_string[]);
void clear_kb(void);
void main(void);
FILE *fp;

void main(void)
{
    char filename[20];
    char rcvd_encoded_string[ENCODED_STR_LEN];
    char rcvd_decoded_string[DECODED_STR_LEN];
    char corrected_output_string[DECODED_STR_LEN];

    int count = 0;
    int i, a;
    int repetitions;
    char ch;
    AJMS cbl;
    unsigned int irq=0, base_address=0;
    char com_port[80] = "1";
    char baud_rate[80] = "4800";
    long baudRate=0;

    printf("Enter name of record file: ");
    gets(filename);

```

```

if ( (fp = fopen(filename, "w")) == NULL)
{
    fprintf(stderr, "Error opening file %s.", filename);
    exit(1);
}
/* printf("Enter comm port to use: ");
gets(com_port);
printf("Enter baud rate: ");
fprintf(fp, "\nEnter baud rate: ");
gets(baud_rate);
fprintf(fp, "%s", baud_rate); */
printf("Enter number of repetitions: ");
fprintf(fp, "\nEnter number of repetitions: ");
scanf("%d", &repetitions);
fprintf(fp, "%d", repetitions);

for (count = 0; count < repetitions; count++)
{
    printf("Cycle number: %d \n", count+1);
    fprintf(fp, "\nCycle number: %d \n", count+1);

    /* initialize variables */
    for ( i=0; i < ENCODED_STR_LEN; i++ )
        rcvd_encoded_string[i] = 0;

    for ( i=0; i < DECODED_STR_LEN; i++ )
    {
        rcvd_decoded_string[i] = 0;
        corrected_output_string[i] = 0;
    }

    while (!irq && !base_address)
    {
        if (strstr(com_port, "1"))
        {
            base_address = 0x3f8;
            irq = 4;
        }

        else
        {
            if (strstr(com_port, "2"))
            {
                base_address = 0x2f8;
                irq = 3;
            }
        }
    }

#define BUFFER_SIZE 4096 /* Use a 4K IRQ Buffer */

    SetupControlBlock(&cb1, base_address, irq, BUFFER_SIZE);

    if (OpenCom(&cb1) != SUCCESSFUL) /* Open Com Port */
    {
        printf("\nCan't Open Com Port!");
        return;
    }

    while (!baudRate) /* Obtain desired Baud Rate */
    {
        baudRate = atol(baud_rate);
    }

    SetBaudRate(&cb1, baudRate); /* Sets up Baud Rate */
    SetDataFormat(&cb1, BITS_8|NO_PARITY|STOP_BITS_1); /* 8-N-1
*/

    if (SetFIFOMode(&cb1) == FIFO_ENABLED) /* Enable 16550 if any */
    {

```

```

        SetFIFOTriggerLevel(&cb1, FIFO_16_TRIGGER); /* Turn on 16 byte RCVR FIFO
*/
        printf("\nUART is a 16550! FIFO's Enabled.\n");
    }
    /* else printf("\nUART is a 16540 or 8250.\n"); */

    if (GetCTSRTSStatus(&cb1) == ENABLED) /* Check if CTS/RTS is on */
        printf("CTS/RTS Hardware Handshaking Enabled!\n");
    else
        printf("\nCTS/RTS Hardware Handshaking Disabled!\n");

    /* printf("\nLoad ANSI.SYS if you wish to display ANSI.\n");
    printf("\nPress [ESC] to Exit this Terminal Program..\n"); */

    for ( i=0; i < ENCODED_STR_LEN; i++ ) /* Loop to fill string */
    {
        if (kbhit())
        {
            ch = (char)getch();
            if (ch == 27)
                return;
        }

        while (CheckQueue(&cb1) != CHARACTERS_WAITING) /* Check IRQ buffer for
chars */
            ;
        rcvd_encoded_string[i] = GetCharacter(&cb1);
        printf(".");

        /* printf("exited while loop! \n"); */
    }

    hamming_decoder(rcvd_encoded_string, rcvd_decoded_string,
corrected_output_string);

    printf("The received decoded string: %s \n", rcvd_decoded_string );
    fprintf(fp, "The received decoded string: %s", rcvd_decoded_string );

    printf("The corrected decoded string: %s \n", corrected_output_string );
    fprintf(fp, "\nThe corrected decoded string: %s \n", corrected_output_string
);

    CloseCom(&cb1);
    DropDTR(&cb1); /* Disable Data Terminal Ready */

    /* printf("we've reached CloseCom and DropDTR. \n"); */
} /* end big for-loop */
printf("Total errors corrected: %d \n", total_errors_corrected);
fprintf(fp, "\nTotal errors corrected: %d \n", total_errors_corrected);
fclose(fp);
printf("just before final brace. \n");

} /* End of Main */

/*Hamming 12/8 Decoder */

void hamming_decoder(char encoded_string[], char error_string[], char
decoded_string[])
{
    unsigned msg_char1, msg_char2;
    unsigned m1a, m2a, m3a, m4a, m5a, m6a, m7a, m8a, p1a, p2a, p3a, p4a;
    unsigned m1b, m2b, m3b, m4b, m5b, m6b, m7b, m8b, p1b, p2b, p3b, p4b;
    unsigned syndrome1, syndrome2, new_syndrome1, new_syndrome2;
    unsigned new_p1a = 0, new_p2a = 0, new_p3a = 0, new_p4a = 0;
    unsigned new_p1b = 0, new_p2b = 0, new_p3b = 0, new_p4b = 0;
    unsigned coded_c, coded_d, coded_e;
    char a_string[2], b_string[2];

```

```

unsigned error_location1, error_location2;
int i, b, n, error_count = 0;
int index = 0;

/* now get the encoded_string, and start decoding it */
/* printf("the encoded_string: %s \n", encoded_string); */

n = strlen(encoded_string);
/* printf("length of encoded_string is %d \n", n); */

if ((n%3) != 0)
{
    fprintf(fp, "\nWARNING: encoded string is not evenly divisible by 3! \n");
    printf("WARNING: encoded string is not evenly divisible by 3! \n");
    if ((n%3) == 1)
    {
        strcat( encoded_string, "  "); /*append 2 spaces*/
        n = n+2;
    }
    else /*n%3 must equal 2, so we only need one space */
    {
        strcat( encoded_string, " ");
        n++;
    }
}
/*end of if clause */

/* parse thru output_string, three characters at a time */
for (i = 0; i < n; i = i+3) /* positions 0 to n-1 */
{
    /* initialize both strings to all zeros. */
    for ( b = 0; b < 2; b++ )
        a_string[b] = 0;
    for ( b = 0; b < 2; b++ )
        b_string[b] = 0;
    /* printf("this is the nulled-out a_string: %s \n", a_string);
    printf("this is the nulled-out b_string: %s \n", b_string); */

    coded_c = (int) encoded_string[i];
    coded_d = (int) encoded_string[i+1];
    coded_e = (int) encoded_string[i+2];

    /* now we have to get 12 bits back out of the 3 chars, which are the
       original message bits plus the parity bits. */
    m1a = m2a = m3a = m4a = m5a = m6a = m7a = m8a = 0;
    p1a = p2a = p3a = p4a = p1b = p2b = p3b = p4b = 0;
    m1b = m2b = m3b = m4b = m5b = m6b = m7b = m8b = 0;

    m8a = (coded_c & 128) >> 7;
    m7a = (coded_c & 64) >> 6;
    m6a = (coded_c & 32) >> 5;
    m5a = (coded_c & 16) >> 4;
    p4a = (coded_c & 8) >> 3;
    m4a = (coded_c & 4) >> 2;
    m3a = (coded_c & 2) >> 1;
    m2a = (coded_c & 1) >> 0;
    p3a = (coded_d & 128) >> 7;
    m1a = (coded_d & 64) >> 6;
    p2a = (coded_d & 32) >> 5;
    p1a = (coded_d & 16) >> 4;

    m8b = (coded_d & 8) >> 3;
    m7b = (coded_d & 4) >> 2;
    m6b = (coded_d & 2) >> 1;
    m5b = (coded_d & 1) >> 0;
    p4b = (coded_e & 128) >> 7;
    m4b = (coded_e & 64) >> 6;
    m3b = (coded_e & 32) >> 5;

```

```

m2b = (coded_e & 16) >> 4;
p3b = (coded_e & 8) >> 3;
m1b = (coded_e & 4) >> 2;
p2b = (coded_e & 2) >> 1;
p1b = (coded_e & 1) >> 0;

msg_char1 = msg_char2 = 0;
msg_char1 = ((m8a << 7) + (m7a << 6) + (m6a << 5) + (m5a << 4) +
(m4a << 3) + (m3a << 2) + (m2a << 1) + m1a);

msg_char2 = ((m8b << 7) + (m7b << 6) + (m6b << 5) + (m5b << 4) +
(m4b << 3) + (m3b << 2) + (m2b << 1) + m1b);

/* Create the string of UNCORRECTED incoming characters. */
error_string[index] = msg_char1;
error_string[index+1] = msg_char2;
index = (index+2);

/*create the syndrome word for the characters received */
syndrome1 = syndrome2 = 0;
syndrome1 = ((p4a << 3) + (p3a << 2) + (p2a << 1) + p1a);
/* printf("This is received syndrome1: %x \n", syndrome1); */

syndrome2 = ((p4b << 3) + (p3b << 2) + (p2b << 1) + p1b);
/* printf("This is received syndrome2: %x \n", syndrome2); */

/* recompute the parity bits from the received characters */

m1a = m2a = m3a = m4a = m5a = m6a = m7a = m8a = 0;
new_p1a = new_p2a = new_p3a = new_p4a = 0;
new_p1b = new_p2b = new_p3b = new_p4b = 0;
m1b = m2b = m3b = m4b = m5b = m6b = m7b = m8b = 0;

m1a = (msg_char1 & 1) >> 0;
m2a = (msg_char1 & 2) >> 1;
m3a = (msg_char1 & 4) >> 2;
m4a = (msg_char1 & 8) >> 3;
m5a = (msg_char1 & 16) >> 4;
m6a = (msg_char1 & 32) >> 5;
m7a = (msg_char1 & 64) >> 6;
m8a = (msg_char1 & 128) >> 7;

new_p1a = m1a ^ m2a ^ m4a ^ m5a ^ m7a;
new_p2a = m1a ^ m3a ^ m4a ^ m6a ^ m7a;
new_p3a = m2a ^ m3a ^ m4a ^ m8a;
new_p4a = m5a ^ m6a ^ m7a ^ m8a;

m1b = (msg_char2 & 1) >> 0;
m2b = (msg_char2 & 2) >> 1;
m3b = (msg_char2 & 4) >> 2;
m4b = (msg_char2 & 8) >> 3;
m5b = (msg_char2 & 16) >> 4;
m6b = (msg_char2 & 32) >> 5;
m7b = (msg_char2 & 64) >> 6;
m8b = (msg_char2 & 128) >> 7;

new_p1b = m1b ^ m2b ^ m4b ^ m5b ^ m7b;
new_p2b = m1b ^ m3b ^ m4b ^ m6b ^ m7b;
new_p3b = m2b ^ m3b ^ m4b ^ m8b;
new_p4b = m5b ^ m6b ^ m7b ^ m8b;

new_syndrome1 = new_syndrome2 = 0;
new_syndrome1 = ((new_p4a << 3) + (new_p3a << 2) + (new_p2a << 1) +
new_p1a);
/* printf("This is new_syndrome1: %x \n", new_syndrome1); */

```

```

new_syndrome2 = ((new_p4b << 3) + (new_p3b << 2) + (new_p2b << 1) +
new_p1b);
/* printf("This is new_syndrome2: %x \n", new_syndrome2); */

/*now check to see if the syndrome words are the same.*/
/* If not, perform bit-flipping on bit in error. */

if ((new_syndrome1 ^ syndrome1) != 0)
{
    error_location1 = (((new_p4a ^ p4a) << 3) + ((new_p3a ^ p3a) << 2) +
((new_p2a ^ p2a) << 1) + (new_p1a ^ p1a));
    /* printf("Error at location %d. \n", error_location1); */
    error_count++;
    switch (error_location1)
    {
        case 1:
        {
            p1a = (p1a ^ 1);
            break;
        }
        case 2:
        {
            p2a = (p2a ^ 1);
            break;
        }
        case 3:
        {
            m1a = (m1a ^ 1);
            break;
        }
        case 4:
        {
            p3a = (p3a ^ 1);
            break;
        }
        case 5:
        {
            m2a = (m2a ^ 1);
            break;
        }
        case 6:
        {
            m3a = (m3a ^ 1);
            break;
        }
        case 7:
        {
            m4a = (m4a ^ 1);
            break;
        }
        case 8:
        {
            p4a = (p4a ^ 1);
            break;
        }
        case 9:
        {
            m5a = (m5a ^ 1);
            break;
        }
        case 10:
        {
            m6a = (m6a ^ 1);
            break;
        }
        case 11:
        {

```

```

        m7a = (m7a ^ 1);
        break;
    }
    case 12:
    {
        m8a = (m8a ^ 1);
        break;
    }
    default:
    {
        printf("something is going wrong in case for syndrome1. \n");
    }
} /* end of switch statement. */

msg_char1 = 0;
msg_char1 = ((m8a << 7) + (m7a << 6) + (m6a << 5) + (m5a << 4) +
             (m4a << 3) + (m3a << 2) + (m2a << 1) + m1a);
/* printf("corrected character is: %c \n", msg_char1); */
}
/* printf("we made it to the end of the if block. \n"); */

b_string[0] = msg_char1;
if (strlen(decoded_string) == 0)
{
    strcpy(decoded_string, b_string);
}
else
{
    strcat(decoded_string, b_string);
}
if ((new_syndrome2 ^ syndrome2) != 0)
{
    error_location2 = (((new_p4b ^ p4b) << 3) + ((new_p3b ^ p3b) << 2) +
                      ((new_p2b ^ p2b) << 1) + (new_p1b ^ p1b));
    /* printf("Error at location %d. \n", error_location2); */
    error_count++;

    switch (error_location2)
    {
        case 1:
        {
            p1b = (p1b ^ 1);
            break;
        }
        case 2:
        {
            p2b = (p2b ^ 1);
            break;
        }
        case 3:
        {
            m1b = (m1b ^ 1);
            break;
        }
        case 4:
        {
            p3b = (p3b ^ 1);
            break;
        }
        case 5:
        {
            m2b = (m2b ^ 1);
            break;
        }
        case 6:
        {
            m3b = (m3b ^ 1);

```



```

        break;
    }
    case 7:
    {
        m4b = (m4b ^ 1);
        break;
    }
    case 8:
    {
        p4b = (p4b ^ 1);
        break;
    }
    case 9:
    {
        m5b = (m5b ^ 1);
        break;
    }
    case 10:
    {
        m6b = (m6b ^ 1);
        break;
    }
    case 11:
    {
        m7b = (m7b ^ 1);
        break;
    }
    case 12:
    {
        m8b = (m8b ^ 1);
        break;
    }
    default:
    {
        printf("something is going wrong in case for syndrome2. \n");
    }

} /* end switch block */

msg_char2 = ((m8b << 7) + (m7b << 6) + (m6b << 5) + (m5b << 4) +
             (m4b << 3) + (m3b << 2) + (m2b << 1) + m1b);
/* printf("corrected character is: %c \n", msg_char2); */
}

a_string[0] = msg_char2;
if (strlen(decoded_string) == 0)
{
    strcpy(decoded_string, a_string);
}
else
{
    strcat(decoded_string, a_string);
}

}

printf("Error count for string: %d \n", error_count);
fprintf(fp, "Error count: %d\n", error_count);
total_errors_corrected = total_errors_corrected + error_count;
} /* End function. */

```

APPENDIX B: PROGRAMMING DIVETRACKER

A fairly complete user's manual is currently included with the purchase of the Desert Star Systems Divetrackers, but some changes in the manual have not kept up with changes in the hardware. According to Marco Flagg, CEO of Desert Star Systems, this situation is being corrected. In the interim, this appendix is intended to provide a step-by-step programming guide for the Divetrackers purchased by NPS for thesis research and incorporation into the Phoenix AUV for navigation and communication.

The Divetracker series components are built using identical microprocessor and modem hardware. The only difference between the DT1-D-S Diver's waterproof station (Figure II-7) and the DT1-Dry surface station (Figure II-8) is the enclosure the units are mounted in, and the way the two units are configured for running the SMARTDIVE application program. The only difference between the DT1-D-S and the DT1-R-S remote unit is the lack of a LCD display and data entry pad. The only difference between the DT1-D-S unit and the DT1-Mod is the DT1-Mod's lack of a waterproof enclosure, pressure transducer, and depth transducer, as it is designed to be mounted in a waterproof ROV or AUV housing. The Divetrackers all utilize flash memory modules to store the programs which they run, whether it be SMARTDIVE, AMODEM, or the supporting programs which provide diagnostics or battery charging functions for the diver's station and the remote units. The non-surface stations (diver's station, remote units) keep all *application* code in FLASH memory, with all essential *configuration* data stored in EEPROM. As both memories are non-volatile, the application is booted and configured automatically as soon as power is applied.

Several concepts regarding the Divetrackers must be understood in order to comply with the following steps. First, the Divetracker unit is incapable of functioning without having program code loaded into it. Prior to use for any application, DiveCode must be loaded into the DiveTracker units and that code configured to run. At the time of this

writing, all DiveCode has been written and compiled by Desert Star Systems. Source code has not been made available to the public, although that will probably change in the future. The existing code has been written mainly to support the navigation and communications functions provided by the basic SMARTDIVE program, as that is what the Divetracker system was originally designed for. However, upon request, code such as AMODEM was written and compiled by Desert Star Systems specifically for this thesis research. All DiveCode is written in C and compiled on an Archimedes DOS/MC68HC11 cross-compiler. One of the reasons that Desert Star Systems wrote and compiled the source code was the high cost of purchasing the Archimedes Cross-Compiler.

DiveCode is loaded into the Divetrackers using a DOS program named DIVETERM.EXE. The process of loading code is outlined in Figure B-1. This file resides in the DTUTILS directory in most implementations of the system. DIVETERM communicates with the DiveTracker using port COM1. Support for other comm ports has not yet been implemented into DIVETERM.

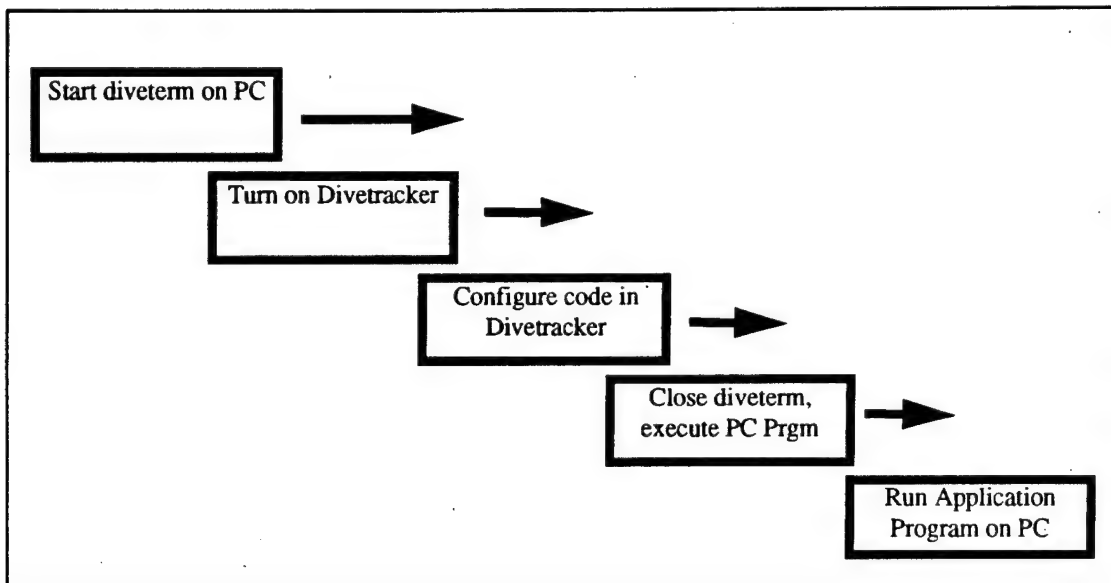


Figure B-1: DiveTracker test procedure.

The test procedure follows. An IBM-Compatible PC is needed which has a free COM1 port as well as having the code which Desert Star Systems supplies with the DiveTracker loaded onto the hard disk. This code comes on 3.5" floppy disks, and is loaded into the computer according to the directions given on the diskettes. The code will install itself into two different directories: DIVEBASE and DTUTILS. Assuming this has been done already, connect the Divetracker's serial connector to the PC's COM1 port, using a DB-25 to DB-9 adapter if needed, and *without* turning on the power to the Divetracker, turn the PC on and let it boot up to the MS-DOS operating system prompt. Change the directory from the root directory to DTUTILS, or if the PC is already on, just change directory to DTUTILS.. Determine the contents of the DTUTILS directory by typing DIR followed by pressing the ENTER key. Verify that the desired DiveCode (.dc suffix) is present in the DTUTILS directory, in the event that that code needs to be loaded. Once identified, write the name of the DiveCode program down for later reference. Execute the DIVETERM.EXE program by typing DIVETERM followed by pressing the ENTER key. The screen shown in Figure B-2 will be displayed.

```
****DiveTerm V1.10****  
  
LINK STATUS: IDLE  
CONNECTED TO: NO CONNECT  
  
Please select:  
  
F1: Select COM Port      F2: Select DiveCode Directory  
F10: Exit DiveTerm      I: Inquire DiveTracker  
X: DiveCode Terminal
```

Figure B-2: Initial DIVETERM screen, DiveTracker OFF.

Now turn on the DiveTracker, either by supplying power to the AUV Module, or by turning on the power switch on the DT1-Dry surface station, or by using the magnetic

pointer in the case of the DT1-D-S Diver's station. The PC display screen should now show the contents of Figure B-3.

```
****DiveTerm V1.10****

LINK STATUS: ACTIVE
CONNECTED TO: DiveTracker model DT1-DRY, s/n 102

MEMORY MAP:
00: a.modem V0.00    01: ***
02: ***              03: <empty>
04: <empty>           05: <empty>
06: <empty>           07: <empty>
08: <empty>           09: <empty>
10: <empty>           11: <empty>
12: <empty>           13: <empty>
14: <empty>           15: <empty>

Please select:

F1: Select COM Port    F2: Select DiveCode Directory
F3: Download DiveCode  F4: Erase FLASH memory
F5: Select DiveCode    F6: Run DiveCode
F7: Set DiveTracker Clock F10: Exit DiveTerm
M: Set Model Type      X: DiveCode Terminal
<: Receive Calibration Data >: Transmit Calibration Data
```

Figure B-3: DIVETERM screen, DiveTracker ON.

The serial number of the DiveTracker will be displayed in the "Connected To" line, as shown above. Depending on the code already loaded into the Divetracker's flash memory, the pages from 00 to 15 may show the name of a program, a series of asterisks ***, or an <empty> sign. Each program takes from 1 to 16 pages of flash memory. Before allowing new code to be loaded into the flash memory, DIVETERM checks to see if enough flash memory is available. If not enough space is available, due to code already in the unit taking up the needed space, DIVETERM will ask whether or not flash memory should be erased. Flash memory is erased in bulk and it cannot be erased one page at a time. Also, if a

program takes up 1 whole page and has one byte left over, the second page it takes up is marked as full also, even though it only has one byte in it.

Depending on what needs to be done by running DIVETERM, whether it be erasing flash memory, downloading code, or simply selecting which code already loaded into flash memory should run when the DiveTracker is powered up, the menu shown in Figure B-3 is where these options are selected. In the example shown, SMARTDIVE could be loaded into pages 03-15 by selecting function key F3. Upon pressing F3, the program will ask for the name of the DiveCode to be loaded. Type the name of the *.dc file written down earlier and press ENTER. The screen shown in Figure B-4 should be displayed. Actually,

```
Specify first FLASH page to be used (0..15): 00

Waiting for ready signal from MCU...
MCU reports FLASH is NOT erased!
Do you want to erase FLASH memory (y/n)? y
Waiting for erase completed...
FLASH is erased!
Now loading page number 0...

Program 7367 bytes, last one at $9fff
Completed 99 percent of transfer...
Page programed OK!

Waiting for ready signal from MCU...
MCU FLASH is erased!
Now loading page number 1...

Program 8771 bytes, last one at $9fff
Completed 99 percent of transfer...
Page programed OK!

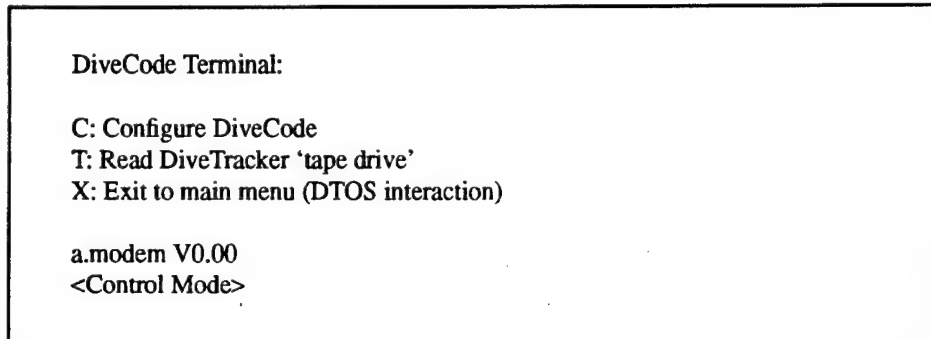
Waiting for ready signal from MCU...
MCU FLASH is erased!
Now loading page number 2...
```

Figure B-4: Erasure of flash memory and code download.

the first section is all that will be displayed, up to the line that reports "MCU reports FLASH is NOT erased!" When prompted, enter a y for YES or a n for NO. If a y is entered, DIVETERM will begin to erase the memory, and when completed, it will begin to load the

code into the flash memory. As it fills each page, it displays a status indicator as shown in the lower half of Figure B-4. Erasure of flash memory takes about 5 minutes and loading code into the flash memory takes about 1 minute per page.

After code has been successfully loaded into the flash memory, it is automatically selected to *run* when the DiveTracker is powered up. However, if DIVETERM is being used to *select* which code is to run, in the event that more than one program is loaded into flash memory, the same steps are taken as described earlier, except that only F5 and F6 need to be pressed to *select* and *run* the desired code. When code is run using F6, or X (capital, as DIVETERM is case sensitive) the following screen shown in Figure B-5 will be displayed.



```
DiveCode Terminal:

C: Configure DiveCode
T: Read DiveTracker 'tape drive'
X: Exit to main menu (DTOS interaction)

a.modem V0.00
<Control Mode>
```

Figure B-5: AMODEM execution response in DiveTerm.

Depending on the application actually running, the DiveCode Terminal will use this screen to display whatever the DiveTracker is outputting through its communications link to the PC, whether it be modem status, as in the case of AMODEM.DC, or baseline range data from the baseline in the case of SMARTDIVE.DC. This display verifies correct operation of the DiveCode.

The DiveCode Terminal is also used to configure the SMARTDIVE application. SMARTDIVE sets various parameters using a *.cfg file which is loaded into the Diver's station and AUV module prior to deployment. This is in contrast to the DT1-DRY surface station which, when running SMARTDIVE, uses a DIVEBASE.PAR file to configure it

when the PC is running the DIVEBASE application. A diagram of this is shown in Figure B-6. Finally, a separate terminal program is used to actually configure AMODEM.DC.

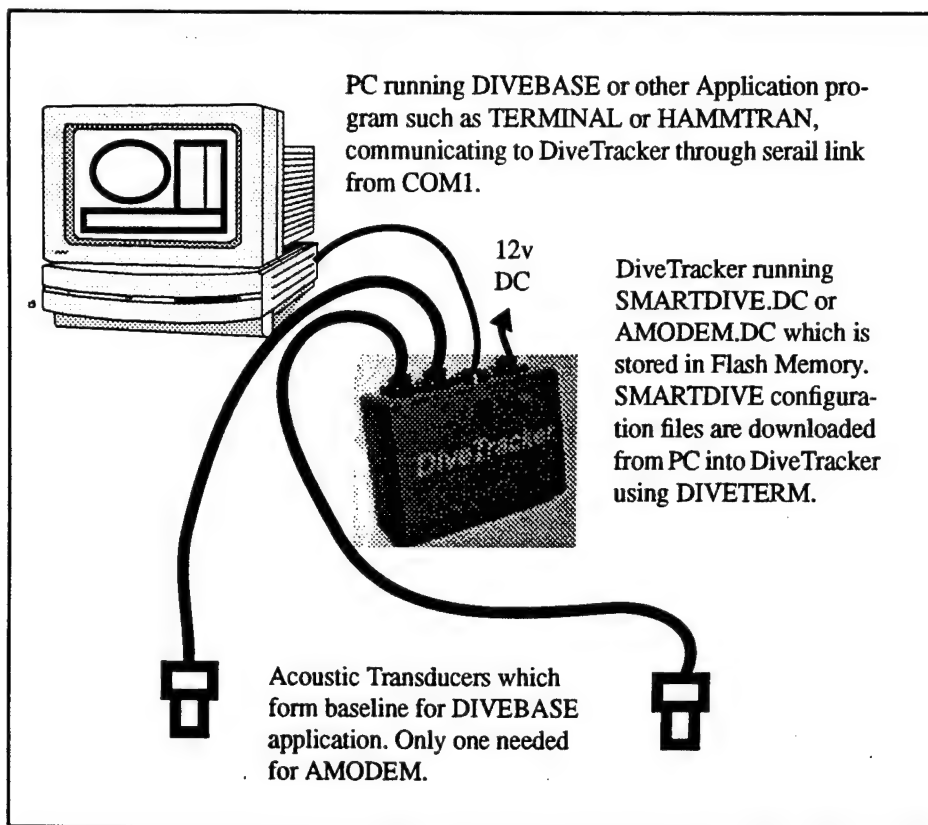


Figure B-6: PC and DT1-DRY Surface Station setup

Any terminal program will work, as long as the communications port and baud rate are able to be set to COM1 and 4800 baud, respectively. AMODEM.DC has two modes of operation: Control and Data. The control mode is accessed by either turning on the DiveTracker when AMODEM.DC is configured to run, or by forcing the DiveTracker to go into the Control mode from the Data mode by pressing the plus key four times (++++). This will not only transmit the four plus signs, but will bring up the Control screen shown in the lower half of Figure B-5. Pressing H (for Help) will display the parameter settings as shown in Figure B-7. The units for each of the settings are described in Figure VI-3. Parameters are modified by pressing the first letter of the desired parameter, such as "S"

for Speed, then a value, using as many zeros as needed to left-justify the value so there are as many digits as there are pound signs (#) in the parameter line. For example, to change pulse length from 1000 to 500, enter "L0500" and press enter. No change will be seen until H is pressed again, and then the new value will be shown. After modification of parameters, the unit can be reset to default values simply by turning it off. When turned back on, it will come up in the control mode. Prior to use as a modem, it must be switched to data mode using a terminal program or the program which will pass data to and from the modem.

Press [ESC] to Exit this Sample Terminal Program..

Use Hayes AT Commands to Talk to your modem. If you don't get a response, try the other COM port, or change your IRQ setting.

Ç

a.modem V0.00

<Control Mode>

H

Speed [S#:0-4]: 1

Power [P###:000-255]: 255

P'length [L####:0100-5000]: 1000

Gain [G#:0-3]: 2

T'hold [T##:0-99]: 16

Filter [F#:0-5]: 0

Checksum [C#:0-1]: 1

Type <D> for data exchange mode

Type <+++> to return to control mode

Type <Q> to switch the modem OFF

D

<Data Mode>

The purpose of computing is insight, not numbers.!

Figure B-7: AMODEM.DC Help screen with parameter display.

APPENDIX C: ACRONYMS

ADATS	Adjustable Diversity Acoustic Telemetry System
AOSN	Autonomous Oceanographic Sampling Network
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Method
AUSS	Autonomous Underwater Search System
AUV	Autonomous Underwater Vehicle
BCH	Bose-Chaudhuri-Hocquenghem (codes)
BER.....	Bit Error Rate
bps.....	bits per second
BPSK	Binary Phase Shift Keying
CPU.....	Central Processing Unit
DES.....	Data Encryption Standard
DLL.....	Dynamic Link Library
DCE.....	Data Communication Equipment
DCT.....	Discrete Cosine Transform
DPSK	Differential Phase Shift Keying
DSP.....	Digital Signal Processing
DTE.....	Data Terminal Equipment
ECB.....	Electronic Code Book
FDDI.....	Fiber Distributed Data Interface
FSK.....	Frequency Shift Keying
IP.....	Internet Protocol
ISI	Intersymbol Interference
ISO	International Standards Organization
ISSM.....	Independent Single Sideband Modulation

kbps..... kilobits per second
 kHz Kilo Hertz
 LAN Local Area Network
 LBL..... Long Baseline (navigation system)
 LASER Light Amplification by Stimulated Emission of Radiation
 LLC..... Logical Link Control
 MABEL..... Modular Abyssal Benthic Laboratory
 (MAC)..... Medium Access Control
 MAN..... Metropolitan Area Network
 MFSK Multiple Frequency Shift Keying
 MLE Maximum Likelihood Estimation
 NIC Network Interface Card
 nm Nautical Mile
 OSI Open Systems Interface
 PC Personal Computer
 PPP Point-to-Point Protocol
 QAM..... Quadrature Amplitude Modulation
 QPSK Quadrature Phase Shift Keying
 RFC..... Request For Comments
 RPC..... Remote Procedure Call
 SBL..... Short Baseline (navigation system)
 SLIP Serial Line Internet Protocol
 SNR..... Signal to Noise Ratio
 TCP..... Transport Control Protocol
 UDP..... User Datagram Protocol
 UWA..... Underwater Acoustic (Channel)
 VHF..... Very High Frequency
 VLF..... Very Low Frequency

WWW..... World Wide Web

APPENDIX D: RADIO ETHERNET BRIDGES

Acoustic Local Area Networks (ALANs) must have an access point - a contact point to a pre-existing network infrastructure, which is usually a higher speed wired or wireless backbone. In the case of this thesis research, the pre-existing network is the Internet, and it can be accessed via an RF link instead of a wired link. The requirement for this type of link is justified simply by the cost and difficulty of running some sort of wire or cable to one of the network nodes which would act as a gateway for the rest of the network.

Normal implementation of a bridge might be on a bouy or float which housed a power supply such as batteries or a solar generator, an antenna, and an acoustic network node with its transducers positioned so as to best communicate with the other submerged network nodes. In the implementations of this type described in the research, a recurring problem was acoustic interference from the buoy anchoring hardware which greatly affected the acoustic node's ability to communicate with its peers. The bridge's RF antenna must be pointed towards the shore receiver site, and arranged so that it maintains direction, a feat not easily acomplished on a bouy.

During this thesis research, two different radio ethernet bridge systems were available for experimentation. Although not utilized specifically with the Divetracker, both systems were successfully used to connect NPS Spanagel Hall to both the NPS Autonomous Underwater Vehicle Laboratory, located at the NPS Golf Course Building 230, and the Hyatt Regency Hotel, located across the highway from NPS as shown in Figure D-1. The former was accomplished in order to provide connectivity to the network on which the NPS Phoenix AUV is operated from the main NPS network. Real-time AUV demonstrations during the Autonomous Vehicles in Mine Countermeasures Symposium were also transmitted back to Spanagel Hall's Computer Science Graphics Laboratory via Multicast Backbone (mBone)[Macedonia, Brutzman 94]. The latter was performed to provide Internet access to the Hyatt Regency Hotel during the Interactive 3-D Graphics Symposium

[Zyda 95]. Both systems performed well, affording connectivity at rates of 800 Kbps and 2 Mbps respectively, which was more than adequate for each application.

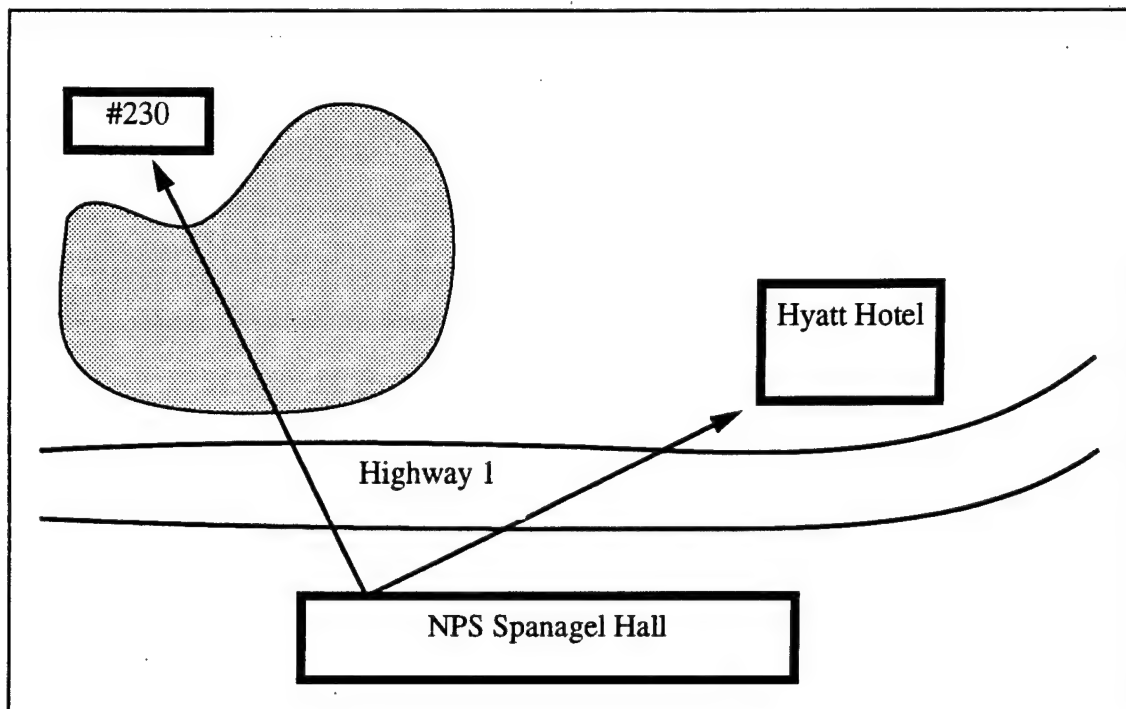


Figure D-1: ARLAN & AIRLAN implementation at NPS.

1. ARLAN 620

The system used first was the Arlan 620, provided by Monterey Bay Aquarium Research Institute (MBARI). This set of two transceivers run on 12 volts, provided by a 120 volt power supply. Each Arlan consists of the following:

- The Arlan unit
- The power supply (120 volt-12 volt adapter)
- Whip antennal
- Yagi antenna and cable

The units accept either thick-wire or thin-wire ethernet. 10Base2 Ethernet (thin) is connected using a BNC connector with a 50 ohm terminator. The third of 3 DIP switches inside each unit switches between thick-Eithernet and thin-wire Ethernet. All 8 switches on SW3 are set to 1 or 0, depending on the type of network media used. Two different

antennas were used: omni-directional whips and directional yagis. The whips were supplied with the unit, and were used for testing and for use within a building. The yagis were used outdoors, and provided enough gain for good connectivity over a range of about 1 mile through two groves of pine trees, which was surprising. The antennas attach to the Arlan via coax terminated with a TNC connector, and it was found that RG-303 MILSPEC cable performed the best when compared to RG-58, which was available in large quantities. RG-303 was not available in large quantities, as the only source seemed to be the Microwave Lab on the 4th floor of Spanagel Hall. The actual impedance of the antenna cable was never determined but was probably 50 or 75 ohms. Lengths longer than 50 feet are not recommended, due to signal attenuation.

In order to setup the Arlan initially, it is necessary to access the setup program of the unit using the DB-8 connector positioned in the bottom of the case. The system setup is retained in flash memory chips and is nonvolatile. Plug a cable with a male DB-8 connector on one end and a DB-9 or DB-25 serial connector on the other end into the Arlan. Connect the other end to either a serial line box or a PC which has the software to run a terminal program. Turn on the serial box, or start a terminal program on the PC, and set the PC's terminal setup to the port you have plugged the cable into. Ensure the cable used is a null-modem cable, not a printer cable. Use of a RS-232 Checker (Light Show) makes it easy to tell if you have the wrong cable- simply attach the light show to the Arlan cable and turn it on, without connecting it to the serial box or PC. Note whether or not TD is lit. Now secure the power to the Arlan, and place the light show on the computer or terminal box, and turn it on. If TD is lit again, you need a null modem or gender changer in the line between the two units. If RTS is lit, you have the correct cable and are ready to continue. Turn the Arlan off, and connect the cable to the serial box or PC, then turn the Arlan back

on. The word "connected" should appear. Press the ENTER key, and you should immediately see Figure D-2 on the serial box or PC.

```
Connected
<TYPE ENTER>
ARLAN 620 V2.01Beta      Main Menu      A620_000db7

  Option      Value      Description

1 - Statistics [ menu ] - Display statistics
2 - Regtable  [ menu ] - Registration table maintenance
3 - Logs      [ menu ] - Alarm and log control
4 - Privilege [ readonly ] - Set privilege level
5 - Close          - Close the telnet session
6 - Help          - Introduction

Enter an option number or name
>
```

Figure D-2: Initial Telnet Screen

The next step is to set the Privilege level, so the setup menu can be accessed. Enter the number 4 at the prompt, and push ENTER. The next screen should look like Figure D-3.

```
Enter an option number or name
> privilege
Enter one of [off, readonly, write] : w
```

Figure D-3: Setting write permission.

The setup program will then display a prompt for the password to be entered. The password is usually known by the last person to use the Arlans, and is also known by Mr. Tom Tengdin at the Monterey Bay Aquarium Research Institute (MBARI). If the password is unknown or cannot be correctly entered, the next method is to reset the whole unit by the following procedure:

- Turn the unit off.
- On the underside of the unit, loosen the two phillips head screws holding the

rectangular plate, and remove the plate. Observe 3 DIP switches: 2 close together, and one set apart from the others a slight amount. Position the unit so the hole with the switches showing is up, and away from you. The DIP switch on the left is SW1. On the middle switch, SW2, set 7 and 8 to both 1. Both switches should already be set to 0 for console port control.

- Turn the power back on, then off again.
- Reset to both switches back to 0. Turn on. The unit should now come up in [write] mode enabled so the password can be reset. Please make a note of the password and tape it into the case so the next person does not have to do this same exercise.

All the options displayed in the brackets [.] need only to have the first letter entered at the keyboard, then press ENTER. The screen shown in Figure D-4 will come up.

ARLAN 620 V2.01Beta		Main Menu	A620_000db2
Option	Value	Description	
1 - Configuration	[menu]	- General configuration	
2 - Statistics	[menu]	- Display statistics	
3 - Regtable	[menu]	- Registration table maintenance	
4 - Logs	[menu]	- Alarm and log control	
5 - Diagnostics	[menu]	- Maintenance and testing commands	
6 - Privilege	[write]	- Set privilege level	
7 - Close		- Close the telnet session	
8 - Help		- Introduction	
Enter an option number or name			
>			

Figure D-4: Arlan screen after entering password.

The most important screen is the menu following selection of option 1 - Configuration. This screen is where the selection is made to modify the network parameters, identification, and radio settings. The screen will look like Figure D-5.

ARLAN 620 V2.01Beta		Configuration Menu	A620_000db7
Option	Value	Description	
1 - Radio	[menu]	- Radio network parameters	
2 - Ethernet	[menu]	- Ethernet configuration	
3 - Ident	[menu]	- Identification information	
4 - Console	[menu]	- Console set-up	
5 - Snmp	[menu]	- Set snmp values	
6 - Dump		- Dump configuration to console	
Enter an option number or name, "=" main menu, <ESC> previous menu			
>			

Figure D-5: Configuration Menu.

Entering a 1 will bring up Figure D-6.

Enter an option number or name, "=" main menu, <ESC> previous menu			
> radio			
ARLAN 620 V2.01Beta		Configuration Radio Menu	A620_000db7
Option	Value	Description	
1 - Sid	[2]	- System identifier	
2 - Bitrate	[860]	- Data bit rate in kilobits / second	
3 - Frequency	[915.0]	- Center frequency in Mhz	
4 - Channel	[11]	- Channel number	
5 - Root	[on]	- Enable root mode	
6 - Linktest	[menu]	- Run a link test	
7 - Extended	[menu]	- Extended parameters	
Enter an option number or name, "=" main menu, <ESC> previous menu			
>			

Figure D-6: Radio Settings.

The setting for option 5 (Root) is different for the other Arlan unit. If the one being set has root mode enabled, the other unit must be disabled, or "off." If this is not done, the bridges will not work. The other options can all be set as before: the unit will display the options and whichever one is chosen will become the new value. However, all the values

shown work well for whatever applications we used them in. Pressing ESCAPE will bring back the original menu shown in Figure D-5. The next menu is Ethernet and is shown in Figure D-7. It is accessed by pressing 2.

```

Enter an option number or name, "=" main menu, <ESC> previous menu
> ethernet
ARLAN 620 V2.01Beta   Configuration Ethernet Menu   A620_000db7
  Option   Value   Description
1 - Active   [ on ] - Ethernet connection active
2 - Size    [ 1518 ] - Ethernet maximum frame size
Enter an option number or name, "=" main menu, <ESC> previous menu
>

```

Figure D-7: Ethernet Menu.

Ensure option 1 is "on," else the Ethernet packets will not be forwarded although the radios link up and communicate between each other. The next screen is most important, as the IP address and Subnet mask is set here. Return to menu shown in Figure D-5 by pressing ESCAPE and press 3 to bring up the Ident menu. The screen should look like the screen shown in Figure D-8.

```

Enter an option number or name, "=" main menu, <ESC> previous menu
> Ident
ARLAN 620 V2.01Beta   Configuration Ident Menu   A620_000db7
  Option   Value   Description
1 - Name    [ "A620_000db7" ] - Node name
2 - Nid     [ 004096000db7 ] - Network address
3 - Inaddr  [ 131.120.063.131 ] - Internet address (YOUR IP ADDRESS GOES HERE)
4 - Inmask  [ 255.255.255.000 ] - Internet subnet mask
5 - Location [ "" ] - SNMP system location
6 - Contact [ "" ] - SNMP system contact name
7 - Date    [ "02/05/94 10:54" ] - Date compiled
Enter an option number or name, "=" main menu, <ESC> previous menu
>

```

Figure D-8: Ident menu.

Options 1 and 2 should not be modified. Options 3 is the IP address which should be obtained from the network system administrator, and must be a unique number. IP addresses cannot be shared. The first 16 bits must conform to the domain, and the 3rd set of 8 bits must identify the network. The last 8 bits identifies the individual node on the network. For some networks, the Inmask must be set as shown above. Not doing so will prohibit telnet operation from outside the local area network. The other fields can be left blank. Press ESCAPE to return to the screen shown in Figure 4, and enter a 4 in order to look at the Console settings as shown next in Figure D-9.

```

Enter an option number or name, "=" main menu, <ESC> previous menu
> console
ARLAN 620 V2.01Beta      Configuration Console Menu      A620_000db7
  Option      Value      Description
1 - Type      [ teletype ] - Terminal type (THIS CAN BE SET TO ANSI, ALSO)
2 - Rate      [ 9600 ]   - Console baud rate
3 - Bits      [ 8 ]     - Bits per character
4 - Parity     [ none ]  - Console parity
5 - Autoconfigure [ off ] - Automatic port configuration
6 - Rpassword                - Set readonly privilege password
7 - Wpassword                - Set write privilege password (DON'T FORGET THE PASSWORD)
8 - Telnet      [ on ]   - Allow telnet connections
Enter an option number or name, "=" main menu, <ESC> previous menu
>

```

Figure D-9: Console menu.

Ensure that option 8 is "on," else the Arlan unit will not allow telnet access, and all access must be done through the DIN-8 plug. Finally, press ESCAPE to return to the

original configuration menu, and press 6 to dump the entire configuration to the screen for viewing. It should look like Figure D-10.

Enter an option number or name, "=" main menu, <ESC> previous menu

> dump

configuration radio sid 2

configuration radio bitrate 860

configuration radio frequency 915.0

configuration radio root on

configuration radio extended sidprefix 2

configuration radio extended Hdrsz default

configuration ethernet active on

configuration ethernet size 1518

configuration Ident name "A620_000db7"

configuration Ident nid 004096000db7

configuration Ident inaddr 131.120.063.131

configuration Ident inmask 255.255.255.000

configuration Ident location ""

configuration Ident contact ""

configuration console type teletype

configuration console rate 9600

configuration console bits 8

configuration console parity none

configuration console autoconfigure off

configuration console telnet on

configuration snmp enabled off

configuration snmp communities add public

configuration snmp communities add proxy

configuration snmp communities add private

configuration snmp communities add regional

configuration snmp communities add core

configuration snmp trapdest none

configuration snmp trapcomm "public"

configuration snmp authtrap off

statistics redisplay_time 10

regtable autoreg on

regtable profile bbnmcst forward

regtable profile radmcst forward

regtable profile bbndst forward

regtable profile raddst forward

regtable profile source off

logs printlevel error/severe

logs loglevel all

logs ledlevel error/severe

logs statuslock off

logs externlevel off

Hit any key to continue ...

Figure D-10: Arlan screen dump.

Little else needs to be set on the units. Each unit when turned on searches for the other. The middle of the three status lights on the top of the unit will blink green continuously until it links with another Airlan which is set to the opposite Root mode; at that time the light will change to steady green. Testing can be done using the whip antennas, or no antennas at all, if the Airlans are in close proximity to each other. Once radio synchronization is achieved, turn the units off and connect the Ethernet cable. Do not forget to use a T-connector so a 50 ohm terminator can be installed if the bridge is the end of the network. Turn the units back on, and when they synch up, they should start passing data.

2. SOLECTEC AIRLAN

The second system used was the Solecetek AIRLAN/Bridge Plus. Like the Airlan, this system operates in the 915 Mhz region, but this system is capable of data rates of 2.0 Mbps, although over a shorter distance than the 6 miles advertised for the Airlans. Three miles is the advertised range with a YAGI antenna, two of which were ordered with the units. Power output is 250 milliwatts, using Direct Sequence Spread Spectrum modulation. Wireless media access control protocol is CSMA/CA, and the network management protocol is SNMP/MIBII.

Two units were purchased for use at the Interactive 3D Graphics Symposium, and are normally in operation between Spanagel Hall (CS Graphics Lab) and the NPS AUV Laboratory at the Golf Course. The Airlan units do not use flash memory to hold their setup data. They are actually IBM compatible 386 computers and boot their configuration from a 3.5" floppy disk. The configuration file is setup using a file called ABCONFIG.EXE which runs on a PC. This executable file reads the configuration off the floppy disk, displays it, and allows modification of the data. The configuration file is then saved back to the disk, and the Airlan unit is rebooted by turning it off and on again. Access to the Airlan is done through a VGA monitor which plugs into the rear of the unit, just like a PC. Power is provided through a standard 120 volt electrical plug. Unlike the Airlans, which could only accept thick and thin Ethernet, the Airlans will accept thick, thin, and twisted

pair Ethernet media (10Base2). The systems have a BIOS just like a PC, and will give a keyboard error when booting up, but this can be disregarded. The antenna connection is for standard cable TV cable, RG-6, and each unit comes supplied with two lengths of cable (15 and 25 feet) and a surge arrester for lightening protection. The antennae supplied are a small omnidirectional for use inside an enclosed room or building, as well as yagi for outdoor use.

Boot diskettes are configured as follows. Copy the file ABCONFIG.EXE to a directory in a PC called AIRLAN. Put the diskette from the Airlan unit in the PC's disk drive and switch to the AIRLAN directory. Start the ABCONFIG.EXE program by typing ABCONFIG and pressing ENTER. Press ALT-F, and the screen in Figure D-11 should be displayed. When the program starts, you want to import the configuration file from the

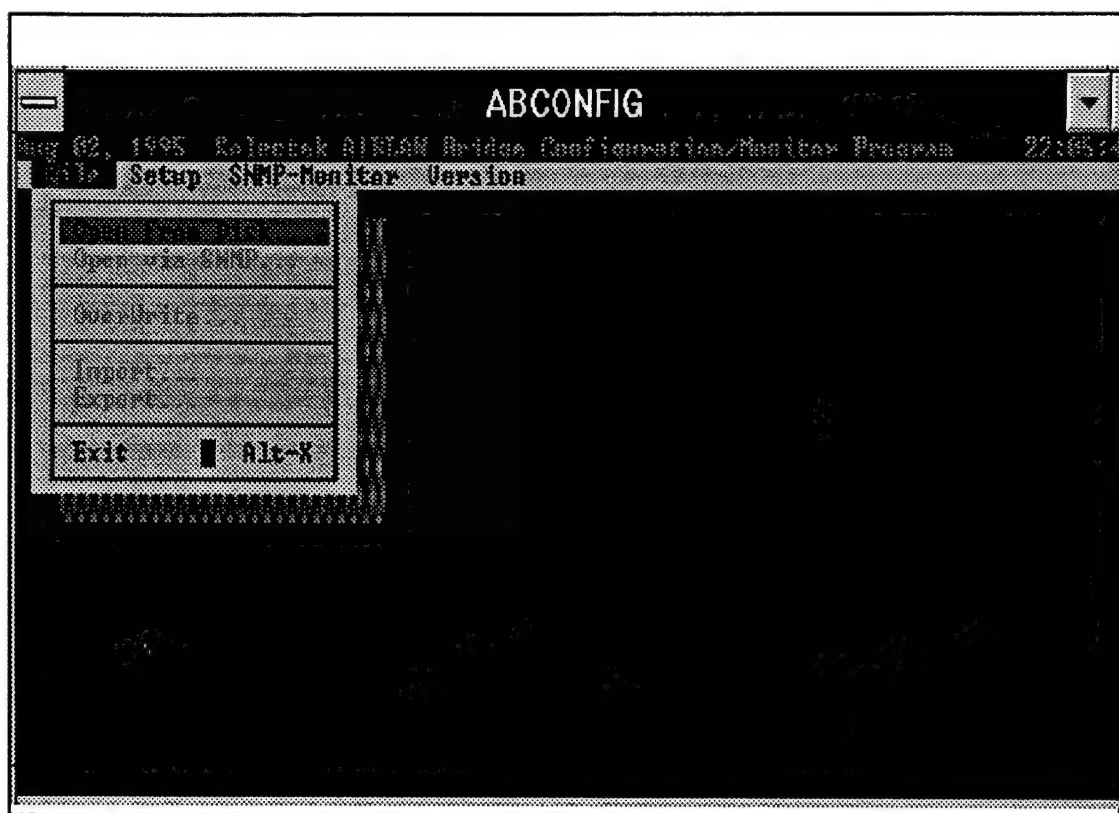


Figure D-11: Airlan Setup Screen

diskette, so type "O", and then next window will ask for the file name. Type the name of the disk drive where the diskette is located, such as "A:," then press ENTER. The program

will search for files with a ".BIN" suffix, and display them. Move the highlighted bar down until it highlights the configuration file you want to use (usually only one) and press ENTER. The program will now import the configuration, and you will see the screen in Figure D-12.



Figure D-12: Setting IP Address in Airlan using ABCONFIG.EXE.

Operation of the Airlan units are essentially the same as the Arlans as far as connection to the network and to the antenna are concerned. The antennae are attached to the highest point possible for the cable to reach and pointed at each other (feed point to the rear, directors laying horizontally active elements both pointing towards the same compass direction) and connected using the 75 ohm antenna cable. Connection of the antenna cable is made to the back of the Airlan on the connector provided. Attach a VGA monitor to the unit using the standard video plug on the rear, and insert a configured boot diskette in the

will search for files with a ".BIN" suffix, and display them. Move the highlighted bar down until it highlights the configuration file you want to use (usually only one) and press ENTER. The program will now import the configuration, and you will see the screen in Figure D-12.

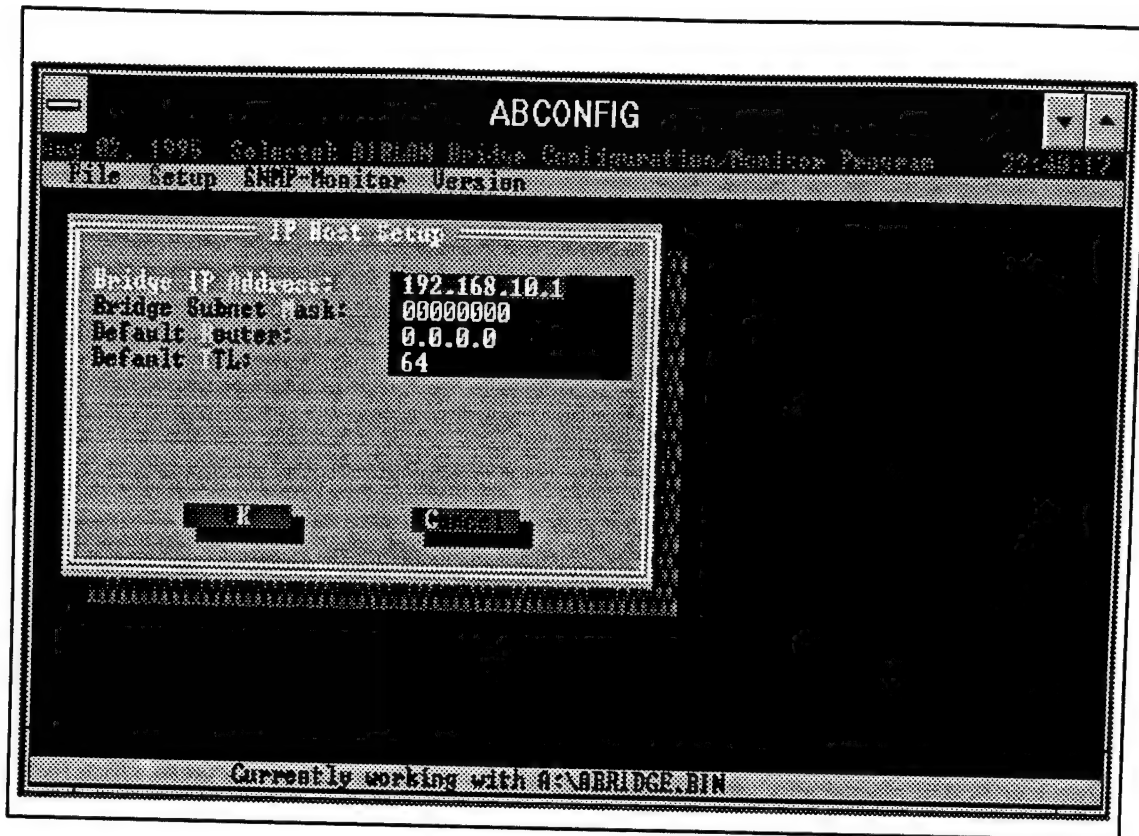


Figure D-12: Setting IP Address in Airlan using ABCONFIG.EXE.

Operation of the Airlan units are essentially the same as the Arlans as far as connection to the network and to the antenna are concerned. The antennae are attached to the highest point possible for the cable to reach and pointed at each other (feed point to the rear, directors laying horizontally active elements both pointing towards the same compass direction) and connected using the 75 ohm antenna cable. Connection of the antenna cable is made to the back of the Airlan on the connector provided. Attach a VGA monitor to the unit using the standard video plug on the rear, and insert a configured boot diskette in the

APPENDIX E: RETRIEVING THESIS AND SOURCE CODE

This thesis is online in both PostScript and HTML format at the following location:

<http://www.cs.nps.navy.mil/research/auv/ipoversw>

Source code used in the thesis is also available at the above address, or at least a symbolic link will be there to the code.

Promodem code is available from:

Eutecnics

c/o Adrian J. Michaud - ProModem!

30 Nagog Pk., Suite 105

Acton, MA 01720

Cost is \$25.00, and purchase price includes all memory models with source code and a printed manual. Microsoft C and Borland Turbo C are supported.

REFERENCES

- Aitken, Peter, and Jones, Bradley, Teach Yourself C in 21 Days, Sams Publishing, Indianapolis, Indiana, 1994.
- Austin, T., "The Application of Spread Spectrum Signaling Techniques to Underwater Acoustic Navigation," *Proceedings of the IEEE Oceanic Engineering Society Conference on Autonomous Underwater Vehicles (AUV) 94*, Cambridge, Massachusetts, July 19-20 1994, pp. 443-449.
- Ayela, Gerard, Nicot, Michel, and Lurton, Xavier, "New Innovative Multimodulation Acoustic Communication System," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. I, pp. 292-295.
- Baggeroer, A.B., "Acoustic Telemetry - an overview," *IEEE Journal of Oceanic Engineering*, vol. 9 no. 4, October 1994, pp. 229-235.
- Bharghavan, Vaduvur, Demers, Alan, Shenker, Scott, and Zhang, Lixia, "MACAW: A Media Access Protocol for Wireless LANs," *Association for Computing Machinery SIGCOMM 94 Proceedings*, London, England, United Kingdom, August 94, pp. 212-225.
- Bessios, Anthony G. and Caimi, Frank M., "Multipath Compensation for Underwater Acoustic Communication," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. I, pp. 317-321.
- Billon, D., and Quellec, B., "Performance of High Data Rate Acoustic Underwater Communication Systems Using Adaptive Beamforming and Equalizing," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. III, pp. 507-512.
- Blahut, Richard E., *Theory and Practice of Error Control Codes*, 1st edition, Addison-Wesley Publishing Company, 1983.
- Blum, Christian, "The Serial Port," Release 18, 22 February 1995, unpublished technical report. Available at ftp://ftp.phil.uni-sb.de/pub/chris/The_Serial_Port
- Brady, D. and Catipovic, J., "An Adaptive, Soft-Decision Multiuser Receiver for Underwater Acoustic Channels," *Conference Record of the Twenty-Sixth Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, California, 26-28 October 1992, vol. II, pp. 1137-41.
- Brady, D., and Catipovic, J., "Adaptive Multiuser Detection for Underwater Acoustical Channels," *IEEE Journal of Oceanic Engineering*, vol. 19, no. 2, April 1994, pp. 158-165.

Brutzman, Donald P., "Encryption Considerations for Autonomous Underwater Vehicle (AUV) Acoustic Telemetry," unpublished paper, Naval Postgraduate School, Monterey California, March 1992. Available at <ftp://taurus.cs.nps.navy.mil/pub/auv/encrypt.ps.Z>

Brutzman, Donald P., *A Virtual World for an Autonomous Undersea Vehicle*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, December 1994. Available at <http://www.stl.nps.navy.mil/~brutzman/dissertation>

Campbell, Joe, *C Programmer's Guide to Serial Communications*, 1st edition, Sams Publishing Company, Carmel Indiana, 1987.

Carey, William M. and Mosely, William B., "Space-Time Processing, Environmental-Acoustic Effects," *IEEE Journal of Oceanic Engineering*, vol. 16 no. 3, July 1991, pp. 285-301.

Catipovic, Josko, "Performance Limitations in Underwater Acoustic Telemetry," *IEEE Journal of Oceanic Engineering*, vol. 15 no. 3, July 1990, pp. 205-216.

Catipovic, Josko A. and Baggeroer, Arthur B., "Performance of Sequential Decoding of Convolutional Codes Over Fully Fading Ocean Acoustic Channels," *IEEE Journal of Oceanic Engineering*, vol. 15 no. 1, January 1990, pp. 1-7.

Catipovic, Josko A., Baggeroer, Arthur B., von der Heydt, Keith, "Design and Performance Analysis of a Digital Acoustic Telemetry System for the Short Range Underwater Channel," *IEEE Journal of Oceanic Engineering*, vol. 9, no. 4, October 1984, pp. 242-252.

Catipovic, Josko, Brady, David and Etchemendy, Steven, "Development of Underwater Acoustic Modems and Networks," *Oceanography*, vol. 6 no. 3, 1993, pp. 112-119.

Catipovic, Josko A. and Freitag, Lee E., "Spatial Diversity Processing For Underwater Acoustic Telemetry," *IEEE Journal of Oceanic Engineering*, vol. 16 no. 1, January 1991, pp. 86-97.

Catipovic, Josko A., Frye, Daniel F., Porta, Dave, "Compact Digital Signal Processing Enhances Acoustic Data Telemetry," *Sea Technology*, vol. 31 no. 5, May 1990, pp. 10-11.

Catipovic, Josko, Johnson, Mark, and Adams, Dennis, "Noise Cancelling Performance of an Adaptive Receiver for Underwater Communications," *Proceedings of the 1994 Symposium on AUV Technology*, July 1994, pp. 171-178.

Chappell, Steven G., Jalbert, James C., Pietryka, Paul and Duchesney, John, "Acoustic Communication Between Two Autonomous Underwater Vehicles," *Proceedings of the IEEE Oceanic Engineering Society Conference on Autonomous Underwater Vehicles (AUV) 94*, Cambridge, Massachusetts, July 19-20 1994, pp. 462-469.

Chen, Kwang-Cheng, "Medium Access Control of Wireless LANs for Mobile Computing," *IEEE NETWORK*, September/October 1994, pp. 50-63.

Cipra, Barry A., "The Ubiquitous Reed-Solomon Codes," *Society for Industrial and Applied Mathematics (SIAM) News*, vol. 26 no. 1, January 1993.

Coatelan, S., and Glavieux, A., "Design and Test of a Multicarrier Transmission System on the Shallow Water Acoustic Channel," *OCEANS 94 IEEE Proceedings*, Brest France, 13-16 September 1994, vol. III, pp. 472-477.

Coates, Rodney, "Underwater Acoustic Communication," *IEEE OCEANS 93 Proceedings*, Victoria, B.C., Canada, 18-21 October 1993, p.III 420-425.

Coates, R., Stoner, R., and Wang, L.S., "The BASS 600 Underwater Acoustic Communication Link," *Sixth International Conference on Electronic Engineering in Oceanography*, Conference Publication No. 394, Cambridge, United Kingdom, 19-21 July 1994, pp. 111-116.

Curtin, Thomas B., Bellingham, James G., Catipovic, Josko and Webb, Doug, "Autonomous Oceanographic Sampling Networks," *Oceanography*, vol. 6 no. 3, 1993, pp. 86-94.

Deaett, Michael, and Audi, Paul P., "Interleaver Performance for FSK Transmission on the Acoustic Fading Channel," *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology (AUV 90)*, Washington DC, 5-6 June 1990, pp. 313-317.

Desert Star Systems, *DiveTracker Users Manual*, Desert Star Systems, Moss Landing, California, July 1994.

Desert Star Systems, "DiveTracker: The One Dive Instrument for All your needs" Specification Sheet, Moss Landing, California, undated.

Datasonics, Inc., "Model ATM-860 Acoustic Telemetry Modem" Specification Sheet, Cataumet, Massachusetts, August 1994.

Datasonics, Inc., "Reliable Underwater Acoustic Data Telemetry" Specification Sheet and brochure, ATM-800 Series, Cataumet, Massachusetts, undated.

Dunbar, R.M., Tennant, A.W. and Wade, S., "A Duplex Laser Location and Communication System for Wideband Real-Time Signaling Between AUVs," *Seventh International Symposium on Unmanned Untethered Submersible Technology (UUST)*, University of New Hampshire (UNH) Marine Systems Engineering Laboratory (MSEL), Durham New Hampshire, September 23-25 1991, pp. 253-263.

Essebbar, A., Loubet, G., and Vial, F., "Underwater Acoustic Channel Simulations for Communication," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. III, pp. 495-500.

Falahati, A., Woodward, B. and Bateman, S., "Underwater Acoustic Channel Models For 4800 b/s QPSK Signals," *IEEE Journal of Oceanic Engineering*, vol. 16 no. 1, January 1991, pp. 12-20.

Feder, Meir, and Catipovic, Josko A., "Algorithms For Joint Channel Estimation and Data Recovery - Application to Equalization in Underwater Communications," *IEEE Journal of Oceanic Engineering*, vol. 16 no. 1, January 1991, pp. 42-55.

Fiorillo, Mark, Irza, John, and Ekhaus, Ira, "Improved Underwater Acoustic Communication For AUVs," *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology (AUV 90)*, Washington DC, 5-6 June 1990, pp. 289-295.

Fischer, Jeffrey H., and Bennet, Kendrick H., "High-Data-Rate, Underwater Acoustic Communication," *Sea Technology*, vol. 34 no. 5, May 1993, pp. 10-13.

Flagg, Marco, "Submersible Computer for Divers, Autonomous Applications," *Sea Technology*, vol. 35 no. 2, February 1994, pp. 33-37.

Flagg, Marco, "Beyond Decompression Computing," *aquaCORPS Journal*, no. N9, pp. 90-92.

Flaherty, Paul, "ECC 1 Reed - Solomon Error Correcting Coder" 9 June 1992. Available at <http://www.baker.com/grand-unification-theory/GNU/software.html#E>

Freitag, LE., and Catipovic, J., "A Signal Processing System for Underwater Acoustic ROV Communication," *Sixth International Symposium on Unmanned Untethered Submersible Technology (UUST) Proceedings*, Baltimore Maryland, June 1989, pp. 34-41.

Galvin, R. and Coates, R.F.W., "Analysis of the Performance of an Underwater Acoustic Communications System and Comparison With a Stochastic Model," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. III, pp. 478-482.

Garner, William R., "An Improved Acoustic Communications System for Autonomous Underwater Vehicles," *1992 Symposium on Autonomous Underwater Vehicle Technology (AUV) 92 Proceedings*, Washinton, DC, USA, 2-3 June 1992, p. 167-174.

Goalic, A., Labat, J., Trubuil, J., Saoudi, S., and Rioualen, D., "Toward A Digital Underwater Phone," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. III, pp. 489-494.

Gray, C.A., Uehara, G.T., and Lin, S., "Bandwidth Efficient Modulation for Underwater Acoustic Data-Communications," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. I, pp. 281-285.

Geller, B., Brossier, J.M., and Capellano, V., "Equalizer for High Data Rate Transmission in Underwater Communications," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. I, pp. 302-306.

Hamming, Richard W., *Coding and Information Theory*, 2nd edition, Prentice-Hall, Englewood Cliffs New Jersey, 1986.

Hedrick, Charles L., *Introduction to the Internet Protocols*, Publication of Rutgers University, 3 July 1987. Available via ftp from <http://www.cis.ohio-state.edu/htbin/rfc/hedrick-intro.html>.

Henderson, G.B., Tweedy, A., Howe, G.S., Hinton, O., and Adams, A.E., "Investigation of Adaptive Beamformer Performance and Experimental Verification of Applications in High Data Rate Digital Underwater Communications," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. I, pp. 296-301.

Herold, D., and Johnson, M., "A Compact Underwater Acoustic Modem," *Proceedings of the IEEE Oceanic Engineering Society Conference on Autonomous Underwater Vehicles (AUV) 94*, Cambridge, Massachusetts, July 19-20 1994, pp. 393-398.

Hoag, David F., and Ingle, Vinay K., "Underwater Image Compression Using the Wavelet Transform," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. III, pp. 533-537.

Howe, G.S., Tarbit, P.S.D, Hinton, O.R., Sharif, B.S., and Adams, A.E., "Sub-sea Acoustic Remote Communications Utilizing an Adaptive Receiving Beamformer for Multipath Suppression," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. I, pp. 313-316.

Internet Architecture Board (IAB), Postel, J., editor, "Internet Official Protocol Standards," Request For Comments (RFC) 1780/STD 1, March 28 1995. Available at <ftp://ds.internic.net/rfc/rfc1780.txt>

Jabbari, Bijan, Editor, "Special Issue on Wireless Networks for Mobile and Personal Communications," *Proceedings of the IEEE*, vol. 82 no. 9, September 1994.

Koehler, Richard L. and Williams, Albert J. III, "Data Direct From the Ocean Bottom to the Laboratory," *IEEE OCEANS 92 Proceedings*, Newport Rhode Island, October 26-29 1992, pp.701-705.

Johnson, Mark, Herold, David, and Catipovic, Josko, "The Design and Performance of a Compact Underwater Acoustic Network Node," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. III, pp. 467-471.

Labat, Joel, "Real Time Underwater Communications," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. III, pp. 501-506.

Luther, G., Vollbrandt, J., and Werner, A., "Main Functions of MABEL Modules and its Individual Use as Separate Scientific Tools," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. III, pp. 587-593.

- Macedonia, Michael R. and Brutzman, Donald P., "MBone Provides Audio and Video Across the Internet," *IEEE COMPUTER*, vol. 27 no. 4, April 1994, pp. 30-36.
- Mackleburg, Gerald R., "Acoustic Datalinks for UUVs," *Sea Technology*, vol. 33 no. 5, May 1992, pp. 10-13.
- McCue, S., Carter, M., and Blidberg, D., "An Investigation into Protocols for Underwater Communications," *Eighth International Symposium on Unmanned Untethered Submersible Technology (UUST)*, 27-29 September 1992, Durham, New Hampshire, pp. 476-488.
- McMullen, Melanie, "An Amazing Aqua LAN," *LAN Magazine*, vol. 9 no. 2, February 1994, pp. 159-163.
- Merriam, Steve, and Porta, Dave, "DSP-Based Acoustic Telemetry Modems," *Sea Technology*, vol. 34 no. 5, May 1993, pp. 24-30.
- National Oceanographic and Atmospheric Agency (NOAA), "Chart 18685: Monterey Bay," Nautical Chart.
- O'Colmain, C., and Fagan, A. D., "Reliable Communications using Trellis Coding in Conjunction with Interleaving and Noise Predictive Decision Feedback Equalization", First International Symposium on DSP for Communication Systems, Warwick, England, Sept 1992.
- Pappas, George, Shotts, William, O'Brien, Mack, and Wyman, William, "The DARPA/Navy Unmanned Undersea Vehicle Program," *Unmanned Systems*, vol. 9 no. 2, Spring 1991, pp. 24-30.
- Peterson, W.W., and Weldon, Jr., E.J., *Error-Correcting Codes*, MIT Press, Cambridge, 1972.
- Piccard, Jacques and Dietz, Robert S., *Seven Miles Down*, G. P. Putnam's Sons, New York, 1961.
- Pless, Vera, *Introduction to the Theory of Error-Correcting Codes*, John Wiley & Sons, New York, 1982.
- Postel, J., "Internet Protocol: DARPA Internet Program Protocol Specification," Request for Comments (RFC) 791, September 1981. Available at <ftp://ds.internic.net/rfc/rfc791.txt>
- Proakis, John G., "Adaptive Equalization Techniques for Acoustic Telemetry Channels," *IEEE Journal of Oceanic Engineering*, vol. 16 no. 1, January 1991, pp. 21-31.

Proakis, John G., Stojanovic, Milica and Catipovic, Josko, "Adaptive Equalization Algorithms for High Rate Underwater Acoustic Communications," *Proceeding of the IEEE Oceanic Engineering Society Conference Autonomous Underwater Vehicles (AUV) 94*, Cambridge, Massachusetts, July 19-20 1994, pp. 157-164.

Proakis, John G., "Coded Modulation For Digital Communications Over Rayleigh Fading Channels," *IEEE Journal of Oceanic Engineering*, vol. 16 no. 1, January 1991, pp. 66-73.

Rockliff, Simon, "rs.c," University of Adelaide, Adelaide, Australia, June 1991. Available at <http://www.nps.navy.mil/research/auv/ipoversw/rs.c>

Romkey, J., "A Nonstandard For Transmission of IP Datagrams Over Serial Lines: SLIP," Internet Engineering Task Force (IETF) Network Working Group, RFC 1055, June 1988. Available at <ftp://ds.internic.net/rfc/rfc1055.txt>

SEACO, Incorporated, *Development of the Acoustic Telemetry System*, Technical Report 2336, Naval Command, Control and Ocean Surveillance Center, San Diego, California, 1992.

Simpson, W., "The Point-to-Point Protocol (PPP) for the Transmission of Multi-Protocol Datagrams over Point-to-Point Links," Request for Comments (RFC) 1331, May 1992. Available at <ftp://ds.internic.net/rfc/rfc1331.txt>

Simpson, W., "The Point-to-Point Protocol (PPP)," Request for Comments (RFC) 1661, July 1994. Available at <ftp://ds.internic.net/rfc/rfc1661.txt>

Stallings, William, *Computer Organization and Architecture*, 3rd edition, MacMillan Publishing Company, New York, 1993.

Stallings, William, *Data and Computer Communications*, 4th edition, MacMillan Publishing Company, New York, 1994.

Stevens, Lou, *A Computer Systems Primer*, unpublished course notes, Naval Postgraduate School, Monterey, California, 2 June 1993, pp. 31-36.

Stojanovic, Milica, Catipovic, Josko A., and Proakis, John G., "Reduced-complexity Simultaneous Beamforming and Equalization for Underwater Communications," *IEEE OCEANS 93 Proceedings*, Victoria, B.C., Canada, 18-21 October 1993, vol. III, pp. 426-431.

Stojanovic, Milica, Catipovic, Josko A., and Proakis, John G., "Phase-Coherent Digital Communications for Underwater Acoustic Channels," *IEEE Journal of Oceanic Engineering*, vol. 19. no. 1, January 1994, pp. 100-111.

Talavage, Jennifer L., Thiel, Timothy E., and Brady, David, "An Efficient Store-and-Forward Protocol for a Shallow-Water Acoustic Local Area Network," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. I, pp. 883-888.

Tanenbaum, A. *Modern Operating Systems*, Prentice-Hall, Inc., New Jersey, 1992.

Tarbit, P.S.D, Howe, G.S., Hinton, O.R., Adams, A.E., and Sharif, B.S., "Development of a Real-Time Adaptive Equalizer for a High-Rate Underwater Acoustic Data Communications Link," *IEEE OCEANS 94 Proceedings*, Brest France, 13-16 September 1994, vol. I, pp. 307-312.

Turner, Elise H., Chappell, Steven G., Valcourt, Scott A. and Dempsey, Martin J., "COLA: A Language to Support Communication between Multiple Cooperating Vehicles," *Proceedings of the IEEE Ocean Engineering Society Conference Autonomous Underwater Vehicles (AUV) 94*, Cambridge, Massachusetts, July 19-20 1994, pp. 309-316.

Urick, Robert J., *Principles of Underwater Sound*, 3rd edition, McGraw-Hill, New York, 1983.

Vukadin, P., and Hodec, Goran, "Acoustic Telemetry System For Underwater Control" *IEEE Journal of Oceanic Engineering*, vol. 16 no. 1, January 1991, pp. 142-145.

Walker, Jerry D., "The 'forward error correction' March," *Unmanned Systems*, vol. 12 no. 4, Fall 1994, pp. 24-29.

Wen, Qian, and Ritcey, James A., "Spatial Diversity Equalization Applied to Underwater Communications," *IEEE Journal of Oceanic Engineering*, vol. 19 no. 2, April 1994, pp. 227-241.

Walton, J., "Evolution of a Search System: Lessons Learned with the Advanced Unmanned Search System," *Naval Command, Control and Ocean Surveillance Center*, Technical Report 1529.

University of Helsinki-Department of Computer Science, *Linux Information Page*, online home page, University of Helsinki, Helsinki, Finland. Available at <http://www.cs.helsinki.fi/linux/>

Zielinski, A., Coates, R., Wang, L., and Saleh, A., "High Rate Shallow Water Acoustic Communication," *IEEE OCEANS 93 Proceedings*, Victoria, B.C., Canada, 18-21 October 1993, vol. III, pp. 432-437.

Zinzow, Mark S., "Modem Basics and Setup," 1 November 1994. Unpublished tutorial, available at <http://ux1.cso.uiuc.edu/~zinzow/modem.txt>

Zyda, Michael, "1995 Symposium on Interactive 3D Graphics," online home page, Naval Postgraduate School, Monterey, CA, 9-13 April 1995. Available at ftp://taurus.cs.nps.navy.mil/pub/SYMPOSIUM_MOSAIC/symposium_final.html

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
Cameron Station
Alexandria, VA 22304-6145

2. Library, Code 013 2
Naval Postgraduate School
Monterey, CA 93943-5101

3. Computer Technology Programs, Code CS 1
Naval Postgraduate School
Monterey, CA 93943-5000

4. Dr. Ted Lewis, Code CS/Lw 2
Chair, Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100

5. Dr. Don Brutzman, Code UW/Br 10
Undersea Warfare Academic Group
Naval Postgraduate School
Monterey, CA 93943-5100

6. Dr. Roberto Cristi, Code EC/Cx 1
Electrical Engineering Department
Naval Postgraduate School
Monterey, CA 93943-5100

7. Dr. James Eagle, Code UW 1
Chair, Undersea Warfare Academic Group
Naval Postgraduate School
Monterey, CA 93943-5100

8. Dr. Anthony J. Healey, Code ME/Hy 1
Mechanical Engineering Department
Naval Postgraduate School
Monterey, CA 93943-5100

9. CDR Michael Holden, USN, Code CS/Hm 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100

10. Dr. Gilbert Lundy, Code CS/Lu 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100
11. MAJ Michael R. Macedonia USA, Code CS/Ma 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100
12. David Marco, Code ME/Ma 1
Mechanical Engineering Department
Naval Postgraduate School
Monterey, CA 93943-5100
13. Dr. Paul Moose, Code UW/Mo. 1
Naval Postgraduate School
Monterey, CA 93943-5101
14. Mr. Lou Stevens, Code CS/St. 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100
15. LT Stephen P. Reimers, USN 2
Department Head School Class 141
Surface Warfare Officers School Command
446 Cushing Road
Newport, RI 02841-1209
16. Dr. Jim Bellingham 1
Underwater Vehicles Laboratory, MIT Sea Grant College Program
292 Main Street
Massachusetts Institute of Technology
Cambridge, MA 02142
17. Dr. Richard Blidberg, Director 1
Marine Systems Engineering Laboratory, Marine Science Center
Northeastern University
East Point, Nahant, MA 01908

18. Dr. Brian S. Bourgeois, Code 7442 1
 Electronics/System Engineer, Advanced Sensor and Survey
 Naval Research Laboratory, Mapping, Charting and Geodesy Branch
 Stennis Space Center, MI 29529-5004

19. Dr. Josko Catipovic..... 1
 Department of Applied Ocean Physics and Engineering
 Woods Hole Oceanographic Institution
 Woods Hole, MA 02573

20. Dr. Tom Curtin 1
 Office of Naval Research (ONR)
 800 North Quincy Street
 Arlington, VA 22217-5000

21. Mr. Marco Flagg, CEO..... 1
 Desert Star Systems
 P.O. Box 6
 Moss Landing, CA 95039-0006

22. Dr. Zygmunt J. Haas..... 1
 Room 4G-520
 101 Crawfords Corner Road
 P.O. Box 3030
 Holmdel, NJ 07733

23. Michael Lee 1
 Senior Research Engineer
 Monterey Bay Aquarium Research Institute (MBARI)
 160 Central Avenue
 Pacific Grove, CA 93950

24. Cabot Management LTD..... 1
 Attn: Wilson E. Russel, President
 700-815 West Hastings Street
 Vancouver, B.C. Canada V6C 1B4

25. Robert Reimers 1
 231 South Elm Street
 Gardner, KS 66030